An object-based codesign methodology.

CAI, Jianming.

REFERENCE

An Object-based Codesign Methodology

Jianming CAI

A thesis submitted in partial fulfillment of the requirements of
Sheffield Hallam University
for the degree of Doctor of Philosophy

February 2001

School of Computing and Management Sciences
Sheffield Hallam University

Abstract

The research into ***Codesign of Hardware and Software*** stems from the development of embedded systems, on which various systems restrictions are imposed. Typical restrictions can be the overall time (latency) to complete an assigned function and the space/power limits within the system. Although software can be used to undertake most tasks in an embedded system, ASIC (Application Specific Integrated Circuits) hardware components sometimes have to be recruited to meet the system constraints. Designing the restricted embedded system with both software and hardware components in it involves the analysis of not only individual hardware/software components but also their mutual influences. Using co-design principles, the approach is to consider both hardware and software from a coherent viewpoint.

This thesis presents the results from our research project in the area of Codesign of Hardware and Software. In this project, we investigated previously published codesign approaches and their methodological supports. The investigation has identified shortcomings and problems with the existing codesign methodologies. A new object-based codesign approach (*Co-PARSE*) is thus developed in this project, which is supported by successive phases, guidelines, and techniques. This methodology offers a coherent design framework for real-time embedded systems and incorporates the criteria of system performance and hardware cost. Tools have been developed to facilitate the use of the methodology. Within the methodology, a high-level system modeling and specification approach has been developed and formalised in the Co-BSL (**Co**design **B**ehavior **S**pecification **L**anguage). The means of transforming Co-BSL specifications to C and VHDL implementations is defined, and a library of VHDL components provided. The thesis documents the partitioning approach taken within the methodology and proposes a new multi-layered bus architecture as a basis for more flexible and efficient implementations. A means of simulating the performance characteristics of this architecture under different configurations is provided, and examples of simulation results are presented. A new embedded system (the Radio Data Computing System) is designed and simulated in the Co-PARSE methodology and simulation results analysed. The thesis concludes with an evaluation of the work carried out in the project and proposals for extending the results obtained in future research.

The major contributions reported in this thesis can be summarised as follows. First, the unified system specification means has been designed, which is embodied in the Co-BSL. It captures overall dynamic aspects and performance constraints in the system under development. This high-level specification language is independent of implementation and does not bias the designer towards the use of hardware or software components at this early stage. Second, within Co-PARSE, the target architecture of the system under development has been exploited to improve the system performance and at the same time to reduce hardware cost. This novel concept has been realised by the introduction of an asynchronous bus protocol and the multi-layer bus communication structure. Third, in order to evaluate the strength and practicability of the Co-PARSE methodology, an extensive case study has been carried out. The new RDC (Radio Dada Computing) System has been designed in the proposed codesign approach. Codesign phases are subsequently applied and the guidelines and tools that are specially developed in support of the methodology are fully utilized.

Acknowledgments

I would like to thank my supervisors Dr. David Lloyd and Dr. Innes Ritchie for introducing me to the exciting research field, *Codesign of Hardware/Software*. Their continuous supervision, guidance and encouragement have been the major energy to push this research project forward. Particularly, my sincere thanks go to Dr. Innes Ritchie for her painstakingly reading through the draft version of this thesis. Her numerous comments and suggestions make this thesis scientifically and technically sound. Overall quality of this thesis has been improved thanks to the insight discussions with Dr. Innes Ritchie and Professor Jawed Siddiqi.

I owe a great debt of gratitude to the Director of my Ph.D. program, Professor Jawed Siddiqi, whose inspiration to my Ph.D. study and the lasting support were the keys to bring this project to a fruitful completion. His comment on this thesis is highly esteemed. I am also grateful to Professor Paddy Nixon and Professor Frank Poole for their valuable helps and censorious comments that corrected and improved the thesis.

Many thanks are due to the past and present senior members and colleagues in Sheffield Hallam University for their valuable support during this project carried out. They include Professor Ian Draffan, Dr. John Travis, Mr. Steve Flowers, Mrs. Linda Harrison, Mrs. Angela Cooper, Ms. Jo Laughton, Dr. Mehmet Ozcan, Mr. Iain Hughes, Mr. Mick Fitzgibbons, and my fellow Ph.D. students in the Hallamshire Business Park.

I am greatly indebted to my wife, Feng-Mei SHEN, and son, Zhen CAI, for their understanding and patience with my hectic research work. Their love and support have accompanied me to the completion of this thesis. I would also like to extend my heartfelt thanks to my parents whose constant stimulation and encouragement were a major source of my determination to complete the Ph.D.

Finally, a *MITRI* bursary from Sheffield Hallam University is gratefully acknowledged, which provided most financial support for this research project.

Contents

Chapter 1

Introduction

1.1 Overview of the Project

This thesis reports on a research project in the area of *Codesign of Hardware/Software*. An object based codesign methodology has been developed, and tools for its use have been constructed. The methodology has been evaluated by means of an extensive case study. The major goals and contributions made in this project are explained in section 1.3 of this chapter, following the introduction to the topic of Hardware/Software Codesign.

1.2 Codesign of Hardware/Software

Computers systems fall into two categories: the general-purpose computer system and the special-purpose computer system. Examples of the former one can be PCs and workstations while the latter one can be industry controllers, automobile controls, medical instrumentation, and so forth. This project is about the research into special-purpose computer systems, which are designed for dedicated applications. As these systems are enclosed in a larger environment, they are often referred to as *Embedded Systems*. The embedded systems can effectively accomplish the tasks that were originally undertaken by other electronics or electronic-mechanical systems. Because of this, the market share in relation to the embedded computer systems constitutes significant increase. According to the statistics [Emb01], 50 to 75 million embedded processors are sold annually and the market for them will grow 30% to 1.2 billion by 2001. The PC microprocessors are only responsible for less than 1% of all processors sold. Embedded processors outsell PC processors by more than 99%. In addition, embedded development tools sales went from $690M in 1997 to $814.7M in 1998 - an increase of 18.1%.

Codesign of hardware/software stems from the development of embedded systems that have various system restrictions, such as the overall time (latency) to perform a given task and the space/power limit. Although software programs can undertake most functions in an embedded system, ASIC hardware components sometimes have to be

employed to meet the system constraints, particularly the overall time (latency) to perform a given task assigned by the larger environment. Most embedded systems consist of application-specific hardware components and programmable components (special or general processors). Those special hardware components are designed to assist programmable components on certain performance-critical tasks. Furthermore, modern embedded systems usually have multiple processors working in a distributed fashion. The embedded system is indeed a complex mixture of hardware and software components. Creating an embedded system that meets those constraints is essentially a hardware and software codesign problem, i.e. the design of hardware and software components that have mutual influences on each other [Wol94].

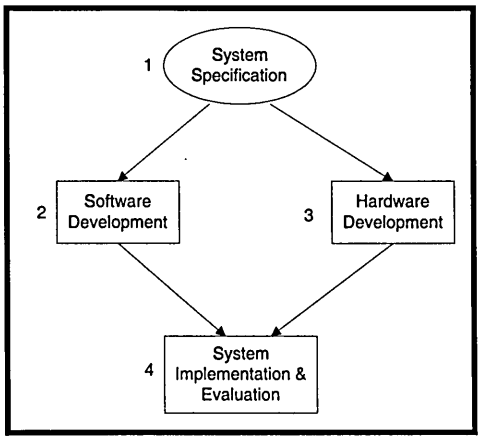


Figure 1.1 Traditional Design Flow

As illustrated in Figure 1.1, the traditional design flow for a computer system is, early in the design cycle, divided into two separate paths that are software (stage 2) and hardware (stage 3) developments respectively. They proceed concurrently without the assessment of mutual impacts and the evaluation of system goals until they reach the final stage (stage 4) that is system implementation and evaluation. This separation is also reflected in the system specification (stage 1), where hardware and software are specified along separate tracks. The assumption behind this practice is that the software designer does not need to be concerned with the low-level hardware details and the hardware engineer, on the other hand, can be relieved from difficulties with understanding complex software design. The separation is mainly in relation to the availability of hardware components and the implementation technology involved. This artificial separation mirrors various technical constraints in the history of computer development. It often results in the designs that exceed both time and budget constraints and yet worse results in failed systems that do not perform as intended.

Theoretically, each application area has its individual, optimal mixture of hardware and software that is best suited to that specific application domain. As computer technology has advanced, there has been a growing interest in the exploitation of combining these two separate developments into a more unified discipline. It has officially been named as "**Codesign of Hardware/Software**" or simply "**Codesign**" [ICSP93].

In fact, the codesign is not an entirely new topic. Computer developers have employed parts of its techniques for many years. The inspiration of this renewed interest is primarily due to the following developments [PF92] [Mic94]:

1. Computing systems deliver increasingly higher performance to end users, which makes special hardware architectures able to assist application-specific software.
2. Architectures with programmable hardware components can now speed up the execution of specific computations or emulate new hardware designs, which enables the designer to trade hardware components in a system for its execution time, development cost, power consumption, and time to market.
3. Significant progresses in hardware synthesis/simulation tools have changed the play field for system designers, which paves the way for the integration of CAD environments in codesign technology.

Since 1993, the international workshop on Codesign of Hardware/Software has been held annually and a large volume of research findings has been published, a number of which can be found in [Buc94]. The codesign has been recognized as a well-established research field.

In contrast to the traditional design flow, Figure 1.2 shows a generic framework for codesign. At the very beginning of the system development, the system is specified by a unified specification means that captures overall dynamic aspects and the constraint requirements in a codesign system. Supported by performance estimation and combined with various system constraints, hardware-software partitioning is carried out in stage 1. Although hardware and software developments are concurrently implemented (stage 2, 3), there can be some feedback and interaction between them during the course of the development. The satisfaction of system constraints can be re-assessed and the system

itself can be re-partitioned. After a mixed system implementation is created, the evaluation in accordance with various system constraints is carried out (stage 4). The whole procedure described above can be repeated until a satisfied codesign system is realized.

In comparison with Figure 1.1, it is evident that the codesign approach maintains the flexibility of choosing best solutions to a specific application domain. The traditional approach only improves software/hardware performance individually so as only to explore very limited design alternatives and then result in a better solution only to that limited part of solution space.

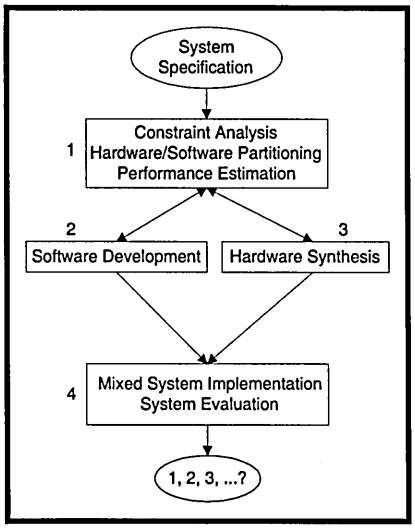


Figure 1.2 Generic Codesign Framework

1.3 Major Goals and Contributions Made in the Research Project

The generation of an inclusive methodology* for codesign is highly complex, but there are many sub-problems. They represent core theories and technologies involved in the research of codesign. They can be yet dealt with individually. These sub-problems are listed as follows:

---

* **NB.** In context we adopt the following definitions by [Cal93].

- A **method** to solve a problem or a technique is characterized by a set of well-defined rules, which leads to a correct solution to the problem.
- A **methodology** has a wider scope than a method. It is a structured and coherent set of methods, guides and tools for determining the way in which a problem can be solved. A methodology leading to the use of techniques can be used to determine whether or not a specific technique is appropriate. It is, therefore, a combination of methods and techniques from various fields. A design methodology is expressed in particular by the progress through successive steps and the tools for efficiently developing a solution to the stated problem and respecting quality criteria.

- Unified hardware/software representation that captures the overall dynamic aspects in a codesign system without bias on hardware or software implementation (*co-specification and modelling)*
- Hardware/software partitioning methods (*design space exploration)*
- Synthesis of hardware/software components and their interfaces *(cosynthesis)*
- Performance estimation and evaluation for a codesign system (*system estimation & evaluation)*
- Applications of codesign technology (*case-studies)*

Since they are too comprehensive to be tackled within one research project, this research has consequently focused specifically on the ***co-specification and modelling***, ***system estimation & evaluation***, ***cosynthesis***, and ***a case study***. The ***design space exploration*** based on the distributed system target architecture is a complicated issue, which involves complex algorithms and system evaluation platforms. Our strategy to attack this problem is to create a feasible evaluation platform. It allows various partitioning schemes to be evaluated on this platform in terms of system performance and hardware cost. Automatic partitioning algorithms/methods, however, have been left over as a future research topic because it is relatively separate from this project. Without the automatic partitioning algorithm, however, the partitioning process has to be operated manually. The profiling technique [EF96] that is widely used in other codesign methodologies has been adopted to collect the information needed in the partitioning phase. In relation to the application area, we assume that the proposed methodology is applied to the realm of embedded real-time system.

In this research, we investigated previously published codesign approaches and their methodological supports. Based on the investigation, we proposed an object-based codesign approach that is supported by a set of methods, guidelines and tools to form a specific object-based codesign methodology. It extensively supports codesign process, including system specification/modelling, hardware/software partitioning, system co-synthesis, and performance evaluation. During the cause of that investigation several experimental case studies are utilized to support the development of these concepts and examples from these studies are given at various places in this thesis. They have also been reported in different publications [CLJ97][CLJ98b].

The methodology originates from the PARSE (*PARallel Software Engineering*) [GJC94] [GGJ95] approach and has evolved to encompass the requirements for codesign. It is therefore named as ***Co-PARSE*** methodology to emphasize its origin.

Major goals of this research are summarized as follows:

- **Development of a system model to capture the overall dynamic aspects of codesign system**

  The model employs object-based and CSP-styled system-level specification notations. It supports hierarchical decomposition, encapsulation and component reuse. Because of its origination from the PARSE approach, it is named "Co-PARSE" and discussed in Chapter 3.

- **Design of a system-level co-specification notation and functional verification technique for codesign systems**

  The notation should be independent of hardware/software implementations. It supports hierarchical decomposition, and promotes staged refinement. By identifying concurrent process objects and their interactions, it captures structural and behavioural properties of codesign systems. The co-specification notation, "Co-BSL", has been designed and described in Chapter 3. The functional verification technique has been developed in Chapter 4, which is established on the conversion from Co-BSL program to VHDL program. VHDL simulations enable the dynamic behaviour of interacting components to be verified at an early stage of the codesign process.

- **A feasible hardware/software partitioning scheme**

  It is restricted to system temporal requirements, improves the system performance, and reduces the hardware cost to a minimum. This scheme is discussed in Chapter 4.

- **Development of new system target architecture and the co-synthesis method for hardware/software interfaces**

  The system target architecture is assumed to be flexible and relatively easy to evaluate in terms of system performance. In addition, template conversions and VHDL packages and libraries should readily support the co-synthesis process. The details of this development are described in Chapter 5.

- **Development of a system-level performance evaluation technique**

It can be used to assess both the performances of hardware/software components and the codesign system's performance. At the same time, the implementation cost is as low as possible. It is also introduced in Chapter 5.

- **Application of the proposed methodology**

  A case study is carried out in order to examine the strength and viability of the proposed methodology. The case study is in favour of embedded real-time system/controller. Its details are included in Chapter 6.

Original contributions have been made in this thesis, which fall into the following categories:

- **Analytical Contributions**
  1. The system-level **C**odesign **B**ehaviour **S**pecification **L**anguage (Co-BSL), which captures overall dynamic aspects of codesign system and its performative constraints (see Chapter 3)
  2. The asynchronous bus protocol and the virtual prototyping technique with the layered bus communication structure, which support the system performance evaluation for codesign system (see Chapter 5)
  3. The co-synthesis method used to implement the hardware/software interfaces, which guides through smooth transition from path definitions in Co-BSL description to the system implementations based on the layered bus communication platform (see Chapter 5 and 6)
- **Developmental Contributions**
  1. Six VHDL packages and one VHDL library that provide a portable platform for future codesign research projects and that are necessary to support the evaluation of the proposed codesign approach
     - Four VHDL packages designed to support, in hardware/software partitioning phase, functional verification and system profiling, which facilitate the template conversion from Co-BSL communication channels to the VHDL simulation program (see Chapter 3)
     - One VHDL package and one VHDL library, which provide communication components and support the communication protocol in the interface co-synthesis phase (see Chapter 5)

- One VHDL package, which implements the major algorithms in the case study, the *Radio Data Computing System* (see Chapter 6)

2. The integration of *ARM SDT Tool Kit* and the *List Scheduling Algorithm* into the system performance evaluation phase (see Chapter 5)

- **Evaluative Contributions**

  Application of the proposed methodology to an extensive case study, the **R**adio **D**ata **C**omputing **S**ystem (RDCS), which evaluates the strength and suitability of the proposed codesign methodology in the development of real-time embedded systems (see Chapter 6)

- **Disseminating Contributions**

  Part of the subject matters addressed in this thesis has been published in the following conference proceedings and technical reports: [CLJ98a][CLJ98b][CLJ97] [CLJ96a][CLJ96b][LJC95][CLJ95a][CLJ95b][CRL00] (see References).

  The publication [LJC95], as a starting point of this project, proposed an object-based codesign approach, which employs PARSE approach including its specification notations for modelling and specifying codesign system. A comprehensive survey of the latest codesign methodologies is undertaken in [CLJ95b] and their possible applications in low-power multimedia system are reported in [CLJ95a]. The literature review in Chapter 2 is largely built upon these two publications. While the publication [CLJ96b] investigated into the high-level specifications for codesign system based on which the Co-BSL is designed, the publication [CLJ96a] examined the feasibility of the conversion from BSL description to VHDL program, which serves as a basis for the conversion from Co-BSL program into C and VHDL program. In addition, two case studies, namely *GSM mobile handset* and *Radio Data System*, have been carried out in this project, which test the viability of co-specification in the Co-PARSE methodology and the functional verification technique in the VHDL simulation. The relevant details are published in [CLJ97] and [CLJ98b]. Due to the lack of specific review on space exploration techniques in codesign society, the publication [CLJ98a] fills this gap by a comprehensive survey of the space exploration techniques. It analyzes a variety of merits on behalf of the five important issues. Part of the content in Chapter 4 is built on this investigation. Finally, the virtual prototyping technique and the

exploitation of system target architecture for system performance proposed in this project has been published in [CRL00].

1.4 Outline of the Thesis

The remaining chapters of this thesis are abbreviated as follows:

Chapter 2 Related Research Works:

The literature review in this chapter takes a close look into the current state of practice in codesign researches and the problems faced by codesign researchers. The codesign methodologies surveyed are well established and all at the leading edge of this active research field. The review has been concentrated on:

- Modelling and specification for codesign system
- Hardware/software partitioning
- Techniques used in system performance evaluation phase
- System target architecture

The review is summarized in a dedicated section that promotes object-orientation in codesign methodology and justifies the proposal of our object-based codesign methodology.

Chapter 3 System Modelling and Functional Verification:

This chapter deals primarily with modelling and specification of codesign system. The modelling technique in the proposed object-based approach is largely built upon the PARSE approach [GJC94] [GGJ95]. Considerable effort is spent on introducing the development of Co-BSL language that is specially designed for capturing the overall dynamic aspects of codesign system. Guidelines for the conversion of the high-level co-specification means (Co-BSL program) to the intermediate-level presentations, i.e. VHDL and C programs, are detailed. The conversion is aimed at preserving the object-oriented features in Co-BSL design, which comprise the encapsulation, communication through message passing, reuse, and scalability. The system functional verification in VHDL simulations with the token-passing protocol and its supporting VHDL packages are also outlined in this chapter.

Chapter 4 Design Space Exploration:

Following the review on design space exploration techniques in the codesign society, the profiling technique specially adopted in this project is presented. The profiling process with VHDL simulations provides important information in the invocation time of primitive process and communication intensity along communication channel. They help identify the time-critical parts and dispatch them to hardware implementation to beat the system time constraints. Possible improvements to the current partitioning method are suggested too.

Chapter 5 The Performance Evaluation:

This chapter is a major part of the thesis. Critical constraints in codesign system are tackled. Although constraints could include system execution time, hardware cost, memory requirement, power consumption, and so forth, we concentrate on the issues related to the system execution time, i.e. system performance, and the hardware cost. The system performance is composed of the performance of individual component and the performance of communication across system target architecture. The performance evaluations for hardware and software components are carried out in both tools and algorithms. While the hardware performance is assessed by the algorithm, namely *List Scheduling*, the software performance is evaluated by using ARM SDT toolkit. In our methodology, the virtual prototyping technique plays a vital role in evaluation of constraint satisfaction. The conceptual system target architecture with the layered system bus structure is proposed and realized in a specially designed asynchronous bus protocol and the VHDL packages/libraries to facilitate the VHDL programming for bus communications. Extra guidelines are also introduced to assist co-synthesis of hardware/software interfaces. The resultant VHDL program is executed in *ModelSim VHDL* simulation environment. Another important issue in this chapter is that the VHDL simulation program is annotated in the performances obtained from the evaluations for individual hardware/software components. The system performance is evaluated in VHDL co-simulation with the annotations. Theoretically, the predicted system performance from this type of VHDL simulation could be accurate to system clock cycles.

Chapter 6 Case Study:

A case study is implemented in this chapter. The case study, namely *Radio Data Computing System,* is used to evidence the feasibility and strength of the proposed object-based codesign methodology. Major codesign phases including system co-specification, system functional verification/profiling, hardware/software partitioning and the performance evaluation are highlighted in the study. Besides, experimental data from VHDL simulations are listed and analyzed. The premier effort are focused on establishing the relations between system latency, hardware cost in conjunction with various partitioning schemes and different numbers of bus layer in the designated system target architecture.

Chapter 7 Conclusions and Further Research Topics:

The thesis is concluded by a summary of the research project together with its progresses and contributions made so far. Possible improvements to this experimental codesign methodology are also suggested. Finally future prospective research directions are addressed.

Chapter 2

Related Research

2.1 Introduction

A number of published codesign methodologies are reviewed in section 2.2. These representative codesign methodologies developed in other institutions are well established and all at the leading edge of this active research area. The emphasis of the following literature review lies in the important issues directly related to three phases in the codesign approach: ***codesign system specification and modelling***, ***hardware/software partitioning***, and ***codesign system performance evaluation***. Another equally important issue we would like to closely examine is ***system target architecture***, which relates to one of the major contributions from this Ph.D. thesis.

In the following sections, each of them (from sub-section 2.2.1 to 2.2.12) deals with an individual methodology. The heading shows its abbreviated name and affiliation. Those without abbreviated names are headed only by their affiliations (sub-section 2.2.7 to 2.2.12). Besides, section 2.3 provides a detailed analysis of the reviewed methodologies and justifies the object-based approach taken in this project. Finally, the object-based codesign approach proposed in this research project is outlined in section 2.4. Its framework is illustrated in a graph. Each phase in the framework is briefly described.

2.2 Review of the Related Research Works

It is worth mentioning at this point the alternative definitions of *co-synthesis* and *codesign*. In some articles, the term *codesign* generally recognizes the difficulty in addressing all of the system design problems in a unified framework while *co-synthesis* concentrates on providing CAD solutions to the sub-problems in codesign system's synthesis [Kum94]. In this thesis, however, we will follow the mainstream definition and do not take account of any particular difference between the term *codesign* and *co-synthesis*.

2.2.1 Approach 1: *Cosyma*, (Germany) Technical Univ. of Braunschweig [EH92] [EHB93][YEBH93] [HE98]

This is a software-oriented codesign approach. The original system is modeled as a software system in C^x language program. The C^x language is a C programming language extended with parallel processes and timing constraints. The system specification written in C^x is compiled into two internal graphs: Extended Syntax Graph (or ES graph) and Basic Scheduling Blocks (or BSBs). A simulator is provided for system functional verification, performance estimation, and partitioning profiling. Those parts (basic block) that are identified as computational bottlenecks and suitable for synthesis of hardware in order to achieve a speedup in overall execution times, are migrated to application-specific hardware (coprocessor), which is then specified in HardwareC. Hardware/Software partitioning is repeated in nested loops. In the inner loop, partitioning is executed by using simulated annealing algorithm, based on cost and timing estimation. The partitions resulted from the inner loop are reexamined in the outer loop that is supported by the hardware simulation tool for run time analysis.

System Specification Means: C^x, an extended C programming language

Partitioning Method: profiling and algorithm plus simulated annealing

Performance Evaluation: co-simulations in both special processor simulator and hardware simulator (Mercury)

Target Architecture: The target architecture is like the one described in Figure 2.1 The general-purpose CPU is implemented in standard Sparc processor and custom device in coprocessor (function unit).

2.2.2 Approach 2: *Vulcan*, (USA) Stanford University [GM93] [GJM94] [Kum94] [Kum96] [AG97] [Mic99]

In contrast to *Cosyma*, a hardware-oriented approach is adopted in the *Vulcan*. The original codesign system is modeled as a hardware system in the hardware description language, *HardwareC*. The *Vulcan* attempts to reduce the cost of its implementation by migrating non-critical operations to a standard processor, such as 8086 or R3000. The partitioning uses heuristics in a cost function with parameters related to the hardware size, processor and bus utilization. While hardware components are synthesized in the *netlist* of logic gates by using *Olympus Synthesis Tools*, software components are compiled into a set of software threads destined for execution on standard processors.

The simulation in the *Poseidon Simulator* evaluates the performance of the final mixed system. The strength of this research lies in the cosynthesis method and tool suite which support the codesign processes including:

- conversion from the top-level HardwareC specification into an internal graph model
- partitioning in the algorithm, based on the internal graph model
- system synthesis particularly for software components and the interface between software and hardware
- low-level simulations in *Poseidon* for system performance evaluation

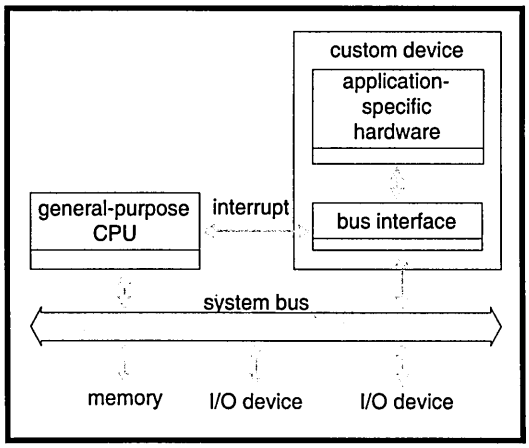


Figure 2.1 Target Architecture of *Cosyma*

System Specification Means: HardwareC

Partitioning Method: algorithm plus heuristics

Performance Evaluation: co-simulations in an event-driven simulator, *Poseidon*

Target Architecture: the target architecture in Figure 2.2 is assumed in this approach. A general-purpose microprocessor is embedded in the system with application specific hardware components. The memory provides storage for program, user data and interface buffer.

2.2.3 Approach 3: *SpecSyn*, (USA) University of California, Irvine [VG92] [GVNG94][GVN94][BG97][GZGH00]

The *SpecSyn* aims at developing a methodology applied to a broad application domain, i.e. embedded systems including codesign system. The main themes in this approach are simulation, rapid prototyping, and framework environment. The system specification is based on the design model, *Program-State Machine* (PSM). A system specification is translated into a hierarchy of program-states in PSM. Each program-state represents a mode of computation and may include standard programming declarations such as

variables, types, and subroutines. In addition to PSM, A VHDL front-end language, *SpecCharts*, is developed to support the captivity of PSM model. This approach also suggests a conceptualization environment, which allows developers to quickly explore and evaluate potential designs. An environment, also named *SpecSyn*, provides users with three types of tool set that demonstrate the initial results: partitioners, estimators, and prototype tools. The design path of *SpecSyn* goes through three major stages: functionality specification, system design, and component implementation. The first stage is to specify the functionality of a codesign system, which is characterized in this approach as executable specification in a machine-readable and simulatable form. The next stage is to map the functionality to system components, which could be memories, buses, ASICs, and processors. The mapping process must satisfy the design constraints such as cost, performance, and power consumption. The final stage relies on the existing hardware/software synthesis and compilation tools.

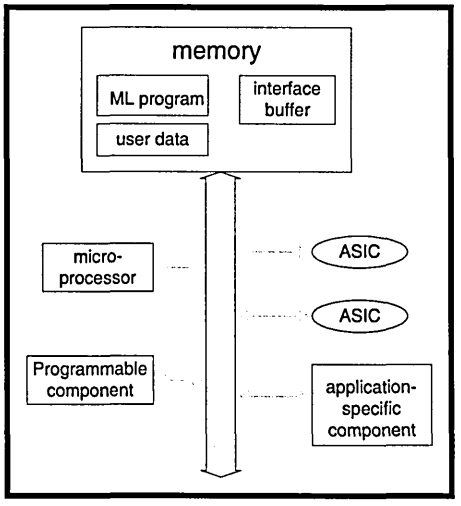


Figure 2.2 Target Architecture of *Vulcan*

System Specification Means: SpecCharts Visual Language

Partitioning Method: clustering algorithm with closeness metrics

Performance Evaluation: In addition to simulations in VDHL for system's functional verification, this approach emphasizes the direct implementations of software in compilers and hardware in synthesis tools (*Executable-Specification Refinement*).

Target Architecture: Although the proposed model has the potentials of using the multiple bus structures and separating the communication constructs from the computation, how different allocation schemes can be examined in terms of the impact on system performance remains as a challenging task.

2.2.4 Approach 4: *Ptolemy*, (USA) University of California, Berkeley [KL93] [KL96] [Tsa00]

The *Ptolemy* is focused on the simulation, prototyping, and software synthesis of digital signal processing systems. The key property in this work is heterogeneity, meaning that software program, hardware modelling and algorithm simulation are embedded in a single design environment. Since *Ptolemy* was developed intentionally for the design of embedded systems with real-time signal processing components, its system specification is encapsulated in an ad hoc formula, particularly tuned into the DSP processing. A Synchronous Data Flow (SDF) Graph is adopted to capture the system model. In this model, an application is specified in a graph, where nodes represent computations and arcs indicate the flow of the data. Supported by the estimation tool, the SDF plus design constraints are divided into hardware (VHDL) and software (C or Assembly) components. Three different tools are provided during the partitioning phase: manual, ILP solver and MIBS. The ILP solver solves problems in an integer linear program while MIBS (Mapping and Implementation Bin Selection) solves the extended partitioning problems in heuristics. The final mixed system is simulated in Ptolemy Environment and implemented in VHDL synthesis tools and software compilers.

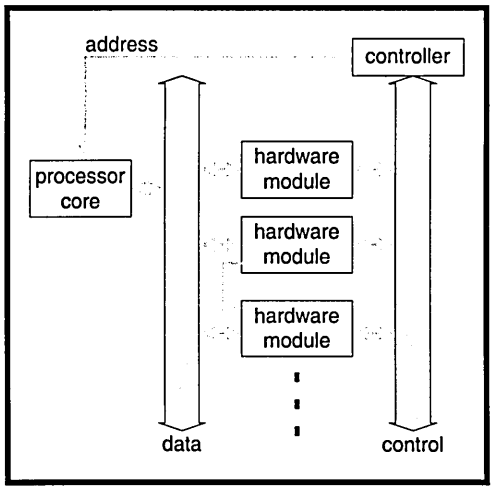


Figure 2.3 Target Architecture of *Ptolemy*

System Specification Means: the synchronous dataflow (SDF) graph

Partitioning Method: experience or algorithm

Performance Evaluation: co-simulations in *Ptolemy* environment or implementation

Target Architecture: Its target architecture is shown in Figure 2.3 [Kal96], which is comprised of a processor core, hardware components (memory, ASICs etc.)

communicating via a single system bus, serial port, or shared memory. This structure is in fact similar to the one in Figure 2.2.

2.2.5 Approach 5: *TOSCA*, (Italy) ITALTEL & Polotecnico di Milano [BFS96] [FS96] [FS99]

The *TOSCA* aims at a codesign environment encompassing all stages in codesign process. It is designed to manage the codesign process particularly for control-dominated applications, such as telecom digital switching subsystems. A codesign system is first specified in the commercial environment, *SPeeDCHART* or *OCCAM II*. The specification is retained in a database that is the hub of all tools included in the environment. Simulations at two levels are provided. The first level is to verify the system functionality and gather profiling information, which will be used during hardware/software partitioning phase. The second level is a VHDL-based virtual co-simulation (the term ***co-simulation*** will be explained in Chapter 5.), which shifts the system tuning from the physical prototyping to the virtual prototyping [BFS94] [AF+97]. Those software threads are converted into a Virtual Instruction Set (VIS) program designed to run on a CPU core while hardware-bound parts are dispatched to hardware components (coprocessors) described in VHDL. The hardware/software interface is synthesized into a system bus that provides data transfer between hardware and software components. The VIS code is created for the purposes of portability and facilitating the VHDL-based co-simulation. Following the system performance evaluation in VHDL co-simulations, the VIS code will finally be retargeted to a real CPU core executing binary code or assembly program.

System Specification Means: SpeedCHART and OCCAM II

Partitioning Method: profiling and algorithm

Performance Evaluation: VHDL-based co-simulations

Target Architecture: A CPU core and a set of coprocessors are interconnected in a single system bus. The target architecture is to be realised in a single-chip. The target architecture is similar to its counterpart in Figure 2.1.

2.2.6 Approach 6: *CODES*, (Germany) Siemens AG [BSV93]

The *CODES* is primarily developed as an integrated hardware-software codesign platform. Its philosophy is to make as much use of relevant existing tools as possible and include as many useful tools as possible. The codesign system is modeled as a set of

communicating *Parallel Random Access-Machine* (PRAM). Its design path is as follows. The original codesign system is specified in *Statemate*^{TM} or *SDL* and is partitioned manually into hardware and software components. The hardware partitions are converted into VHDL program and software partitions into C program. Both C compiler and VHDL synthesizer are used to create the executable code and the netlist that can be further processed in other hardware design tools for placement and routing. The final result is a physical prototype for performance evaluation.

System Specification Means: Statemate^{TM} or SDL

Partitioning Method: experience (manual operation)

Performance Evaluation: simulations in Statemate^{TM} or SDL tools and co-simulations on the physical prototyping

Target Architecture: The target architecture assumed in this approach is a structure composed of a processor, memory, off-the-shelf components and some ASICs, which is similar to the target architecture in Figure 2.2.

2.2.7 Approach 7, (USA) Carnegie Mellon University [TAS93] [PTWP99] [PPT00]

Its codesign system is modeled as the system with communicating sequential processes (CSP) [Hoa85]. This approach concentrates on two important codesign issues: co-simulation and system cosynthesis. The hardware simulation tool (*Verilog* simulator) is connected to the UNIX software processes via BSD UNIX socket facility, which enables co-simulations that verify the functionality of mixed hardware-software descriptions and supply a part of the information with regard to the system performance. The system cosynthesis modifies the hardware-software partitions and control concurrency to make the target system's behaviour meet the design goals. Unlike other methodologies, its partitioning phase takes place at the task level, which consists of a sequence of operations abstracted from a process. The advantage of this treatment will be discussed in Chapter 4. In addition, a physical prototyping system is developed as a testbed, providing a precise and flexible assessment of the system performance.

System Specification Means: Verilog and other UNIX-based software programming languages

Partitioning Method: experience plus the information from high-level software simulation

Performance Evaluation: co-simulations by using the system development board (physical prototyping)

Target Architecture: The target architecture is same as in Figure 2.1. It consists of a general purpose CPU running on an operating system and an ASIC communicating with the CPU via interrupt-driven I/O or with memory and other I/O devices through system bus. The memory and I/O devices are attached to the system bus.

2.2.8 Approach 8, (USA) University of California Berkeley [CGJ+94]

This is a rigorous codesign framework that supports the development of codesign system in synthesis, optimization, and verification. Its system model, *Codesign Finite State Machine* (CFSM) is an extension of classical *Finite State Machine* (FSM). The codesign starts within a unified framework that is unbiased towards the final implementations. The programming language, *Esterel*, is used for system specification that is then translated to CFSMs. The system is then interactively partitioned into hardware and software components. This approach can synthesize the entire design, including the hardware-software interface. Due to its FSM root, synthesizing hardware partitions into a combinational circuit is a natural procedure, whereas software partitions have to be converted into the portable C code via an intermediate model called *software graph* (S-graph). The FSM model derived from CFSM is compatible with the input format in many formal verification algorithms. This feature provides the possibility of mathematically verifying the design in the early stage of codesign process. Another key feature of this approach is the transition from specification to implementation, which is achieved through the maintenance of the finite state machine model throughout. Because of the obstacle of state explosion, this approach is suited to small control-dominated embedded systems.

System Specification Means: Esterel language (real-time software specification language)

Partitioning Method: experience (manual operation)

Performance Evaluation: formal verification on CFSM model plus the simulation in intermediate internal format to complement the verification for some special cases

Target Architecture: This approach has focused on the internal theoretical model, CFSM. Existing software/hardware design tools especially for the development of real-time embedded system have been used in other codesign phases. The system

architecture is not quite clear, but mostly like the one with single bus system plus other communication mechanism such as interrupt and other hardware links, many of which are commonly used in the current real-time embedded system.

2.2.9 Approach 9, (USA) Princeton University [WDW94] [YW95]

In contrast to other approaches, this research concentrates on the co-specification method, whilst its ultimate aim is to develop an automated partitioning tool in the hardware/software partitioning phase. A great deal of effort has been spent on the framework, which proceeds from the object-oriented co-specification to other codesign phases. At the system level, a prototype language, *Object-Oriented Functional Specifications* (OOFS), is used to describe the system objects and operations within an embedded system. The codesigner first has to divide the specification in software, hardware, or codesign parts. Pure hardware or software parts are then converted into C++ classes, whereas codesign parts are compiled into Bestmap-C code for hardware and C++ classes for software. The performance and cost for hardware parts can be obtained from Bestmap-C synthesis and simulations.

System Specification Means: object-oriented functional specification (OOFS)

Partitioning Method: manual operation (experience) and facilitated by the information from simulations in C++ and Bestmap-C

Performance Evaluation: co-simulations in C++ program and Bestmap-C simulator

Target Architecture: Although the codesign target architecture could theoretically be any distributed structure, the example of the target architecture demonstrated in relation to the performance evaluation in this approach is made up of a host microprocessor and ASICs, connected to a system bus. This is rather similar to the architecture in Figure 2.2.

2.2.10 Approach 10, (UK) UMIST [Edw93] [EF94] [EF96] [EFW97]

The contribution of this approach is an integrated codesign environment suitable for general-purpose applications, rather than the more domain-specific approaches taken by other researchers. A software-initiated approach is adopted, where the whole system is treated as a software system written in C. Supported by an interactive profiling tool that identifies performance critical regions in the original system, the C program is subsequently partitioned into software and hardware modules. The critical regions are

implemented in custom hardware. The system development board with FPGAs enables accurate and flexible evaluation of the system performance.

System Specification Means: C programming language

Partitioning Method: profiling to detect the computational bottlenecks

Performance Evaluation: co-simulations on the development board (physical prototyping)

Target Architecture: As illustrated in Figure 2.4, the twin bus architecture is used for codesign system development. The processor P is a 16MHz i960 and the M is composed of 256 KB static read-write memory. I/O consists of a single serial port. The custom hardware is a Xilinx 3090 FPGA for programmable hardware implementation. Since the AT bus and the interface are only used for communications between the development board and a PC host, the target architecture is in fact similar to the one in Figure 2.2.

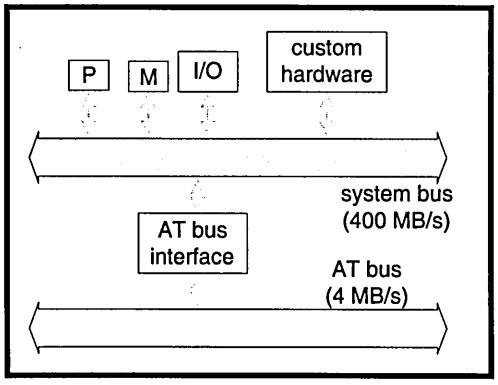


Figure 2.4 Development Board Architecture & Communication with Host PC

2.2.11 Approach 11, (USA) University of Virginia [KAJW93] [KAJW96]

This research outlines a preliminary framework for codesign. The following features are characteristic of this approach:

- integrated codesign process
- model continuity
- exploration of hardware/software tradeoffs
- evaluation of hardware/software alternatives

The codesign system is specified as a set of VHDL concurrent processes, which communicate in certain designated fashion. The estimation and evaluation of system performance are supported by VHDL simulations that are based on the *(un)interpreted modelling methodology* [Ayl92]. Since VHDL provides a behaviour description of

hardware from system level down to the gate level, the partitions for hardware can be implemented in a straightforward manner by hardware synthesis tools.

System Specification Means: VHDL/Petri nets specification

Partitioning Method: profiling and experience (manual operation)

Performance Evaluation: distribute-event simulation in VHDL program with the token passing protocol

Target Architecture: Distributed system architectures are assumed in this approach.

2.2.12 Approach 12, (Sweden) Royal Institute of Technology [JE+94]

The novelty of this approach is its fully automatic hardware/software partitioning and memory allocation achieved by linking the GNU CC compiler to a behavioural VHDL generator and high-level synthesis tools. The compiler is invoked three times. The first is to make the profiling marks in the specification described in C programming language and the second invokes an estimator to calculate the speedup factor under hardware implementation and identify program regions suitable for hardware implementation. The final one generates assembly and VHDL codes for software and hardware respectively. The partitioning problem is formulated, as finding a subset of the program regions suitable for hardware implementation and it is possible to gain the greatest system speedup as they can be fixed into the hardware limit in terms of logic gates. The dynamic programming technique is used in searching for the best subset.

System Specification Means: C or C++ specification

Partitioning Method: profiling and algorithm plus dynamic programming

Performance Evaluation: co-simulations in physical prototyping

Target Architecture: Its target architecture is a board with a microprocessor, ASICs, FPGAs, standard components, and memory. All of them communicate through a single system bus. This architecture resembles the one in Figure 2.2.

2.3 Analysis of Existing Approaches

The codesign methodologies surveyed above are then analyzed, with the focus on the following issues:

- Codesign system model and specification means
- Use of VHDL in codesign approaches
- Hardware/software partitioning method

- Performance evaluation
- System target architecture

The outcome of this analysis shall support the objectives set out in our research project and leads to an experimental object-based codesign framework proposed at the end of this chapter.

2.3.1 Codesign System Model and Specification Means

The system model plays an important role in the codesign. It is particularly significant due to the increasing complexity and the decreasing time to market. The importance of a system model is apparent but often overlooked, particularly when dealing with small-scale systems. A system model can provide both the detailed understanding of system behaviour and the transformation capability that allows the generation of design alternatives. A system model's ability to extend across different system development phases is essential for the validation of system-level models and their hardware/software implementations.

As the survey shows, the following models are involved:

- CSP (approach 5 and 7)
- Petri nets (approach 11)
- Codesign Finite State Machine (approach 8)
- Program-State Machine (approach 3)
- Parallel Ransom Access-Machine (approach 6)
- Others (approaches 1, 2, 4, 9, 10 and 12)

The system specification means adopted by those codesign approaches examined above are categorized as follows:

- C or C-type, C++, and Esterel (approaches 1, 8 and 12)
- SpecCharts (approach 3)
- VHDL, Verilog, and HardwareC (approaches 2, 7 and 11)
- SpeedCHART, Occam II, and Petri nets (approach 5)
- Statemate^{TM}, SDL, and SDF graph (approach 4 and 6)
- OOFS (approach 9)

Current practice in relation to the codesign system modelling and specification is in an ad-hoc state as it depends on development tools (environments) and, also, experiences available to the developer. Those models and specification being used by other researchers were originally designed solely for hardware or software system development. They are either hardware or software implementation-biased. Therefore, the research into their adaptability in codesign methodology needs to be highlighted.

2.3.2 Use of VHDL in Codesign Approaches

A number of codesign approaches employ HDLs or VHDL [IEEE94] [IEEE98] as system specification means or the intermediate description tools that are directly linked to the hardware synthesis tools. Compared with other HDLs, VHDL has the following advantages: [Per94]:

- VHDL is an IEEE standard used as an interface between humans and design automation tools.
- Many different design methodologies and design technologies are supported by VHDL.
- It is independent of both technology and process.
- VHDL supports behavioural description of hardware from system level to gate level.
- Its philosophy is similar to that of many modern programming languages so that it is well facilitated by design decomposition aids (e.g. packages, configuration declarations and the concept of multibodies).

Due to these advantages, VHDL deserves to be emphasized as a design description tool in codesign and its development environment should be extended to support codesign process.

2.3.3 Hardware/software Partitioning Method

Hardware/software partitioning is a really challenging task in codesign research. The major problem is due to the contradictory dependency, where an individual partitioning scheme exerts a great impact on the system performance, but the partitioning process itself relies on the outcomes from the evaluation of system performance after the partitioning. A review of the previous research in this area has been published in

[CLJ98a]. Here we are only concerned with the following three aspects: partitioning method, partitioning process, and partitioning input.

The partitioning methods supported by the above-surveyed codesign approaches are classified as algorithm, experience, and profiling. In other literature [GVNG94], these terms are referred to as deterministic, statistical, and profiling. The deterministic approach requires that all data dependencies are removed and all costs of components known. It can lead to an effective partitioning, but it can fail when those elements are unavailable. The statistical approach is based on the analysis of similar systems and design parameters. The profiling approach is straightforward, and generally yields better results because the partitioning can be determined even when strong data-dependency exists. The deterministic approach requires that intervention from the designer is minimal, which could lead to an automated partitioning process. From this point of view, approaches 1, 2, 3, 4 and 12 represent the most sophisticated partitioning methods while approaches 5, 10 and 11 need statistical information from analysis of similar systems and/or designer's experiences.

A partitioning process can be either intricate or straightforward. Approach 1 introduces the idea of two partitioning loops. The inner loop iteration relies on merit estimation and software simulation to meet the time constraint. By merit estimation we mean that the system performance should be assessed against system constraints established during specification phase. This assessment could be the emulation on a prototype system or software simulations. Because at this stage the real system is not yet created the precision of assessment varies due to the methods adopted. The outer loop checks the integrated system's performance. It uses the *objective function* and the *simulated-annealing* partitioning algorithm. The partitioning can be iterated in either the inner loop or outer loop. Approach 2 uses an objective function that incorporate the metrics of hardware size, program/data storage, bus bandwidth, data rates, synchronization overhead, and the period of time between certain operations. It adopts the partitioning algorithm developed by other researchers to guide the partitioning process. Approach 3 supports various closeness metrics to group behaviours for execution on system components that are chips, blocks on a chip, off-the-shelf processors, memories and

buses. More partitioning mechanics have been published in approach 1, 2, and 3 than other studies and more experiences are presented in these approaches.

Although hardware/software partitioning can be performed at different abstraction levels (i.e. *partitioning granularity*), it mostly occurs at behavioural or structural levels. All the partitioning techniques described above could support behavioural level partitioning. The partitioning granularity varies from task level to single statement level. They can be roughly divided into two categories: *coarse-granularity* (task, function, and process level) and *fine-granularity* (statement block and single statement level). Accordingly, approaches 1, 2, 3, and 9 are fine-granularity and the others are coarse-granularity. Partitioning with coarse-granularity is a common practice in the manual partitioning operation. It implies larger chunks of functionality enclosed in a partitioned component and less communication overhead across the components. In contrast, partitioning with fine-granularity includes more decomposed objects and heavy communication overheads. Therefore, careful consideration has to be taken in order to balance these elements.

2.3.4 Performance Evaluation

Performance evaluation generally takes place after partitioning and/or the interface co-synthesis phases. Related techniques employed to date in the codesign approaches can be characterized as follows:

- Implementation
- Physical prototyping (approaches 3, 6, 7, 10 and 12)
- Virtual prototyping and co-simulation in software (approaches 1, 2, 4, 5, 8, 9 and 11)

Some of the approaches (such as approaches 3, 4 and 6) also employ implementation as one of the options in their evaluation stage apart from the prototyping techniques.

Because of its inflexibility and huge cost, the evaluation by the direct implementation of a codesign system is less attractive. Physical prototyping is precise but it is costly and less flexible. Sometimes, it is virtually impossible, particularly on occasions when some of the system components are not available. Although virtual prototyping is less accurate, it costs less and easy to operate. It is also very effective in the cases where the performance evaluation is undertaken on changeable system architectures. Due to these

advantages, there has been a growing interest in virtual prototyping and the research to improve its simulation accuracy (i.e. speed, precision, facility, and so forth) has intensified [BFS94] [BE97] [PLCV97] [HB97].

2.3.5 Target Architecture

In spite of its extra connection to the CPU via an interrupt device, the target architecture in Figure 2.2 is not fundamentally different from its counterpart in Figure 2.1 from the system architecture's point view because they both use a single system bus for communication between hardware and software components. Yet, these two architectures have been taken as the orthodox system target architectures. The reason they are so popular is probably because of their simplicity that makes the co-synthesis of hardware/software interface and performance estimation/evaluation convenient.

On the single bus platform, the general-purpose processor is naturally taken as a bus master that controls bus traffic, so that the application-specific hardware component can be simplified as a bus slave without the bus traffic control facility. The inclusion of such functionality in an ASIC chip would significantly increase the total hardware cost and complexity. In addition, the single level memory subsystem avoids the complexity that would otherwise arise when analyzing and synthesizing hierarchical memory subsystems.

A few of the approaches do allow designers to specify a codesign system with a distributed system model, but there has been very limited research results reporting on the allocation or mapping of the high-level specification into low-level distributed target architectures i.e. *the components connected to the system bus are self-clocked and the system communication path is configurable.*

2.3.6 Conclusions from the Analysis

The characteristic of a codesign system is regarded as increasing complexity and decreasing production time. For example, the complexity scale of a mobile terminal will be between 500,000 and 1 million transistors and industrial design times allowed for this kind of applications are typically less than one year [GG94]. The transistors for digital signal processing in the digitized camcorder are around 0.37 million [TA92] [SC94]. Furthermore, in addition to ASICs, microprocessors and programmable

components are increasingly included, which makes these systems a complicated mixture of hardware and software components. To address the system-level design problems, new design methodologies are being developed. They require unified system descriptions that allow the developer to express and evaluate alternative partitioning strategies within the same notation [JDV92], supported by mechanisms for the expression of design constraints at a high-level. The object-based codesign methodology that provides system descriptions independent of the hardware/software implementation aspects, supports hierarchical decomposition, and promotes staged refinement should be explored.

Finally, each application domain may require a special target architecture that is best suited in that area in relation to constraints of hardware cost and system performance. Current codesign research is overwhelmingly based on the fixed system target architecture that is featured as the system target architecture with a single bus structure. Specialized hardware components attached to the single bus can certainly reduce the system execution time, but the codesign system based on this architecture inevitably suffers from the communication bottleneck inherited from this type of target architecture. This pessimistic view is reflected in [Edw97]. It instead suggests that the *Application Specific Instruction Processor* (ASIP) would benefit the system's execution more than ASICs do. While ASIPs remain as a promising option, we would argue that the *distributed system target architecture* could also be considered as an alternative solution.

As an example, a distributed target architecture is depicted in Figure 2.5 [SB91] [Sri93]. It is organised in a structure with 4 layers. The bottom two layers are extended by the custom boards, each having one or more programmable processors. Each processor in turn coordinates a number of application specific slave modules which can be either hardware or software components. This target architecture exercises the hierarchical bus organization that increases the communication bandwidth. On the other hand, the system performance evaluation of this architecture has relied on the physical prototyping. While this type of system architecture provides flexibility and scalability, the system performance evaluation and partitioning strategy present a genuinely challenging task. Physical prototyping is no longer feasible and costs increase too much

when the system structure changes. In addition, the hardware/software partitioning strategy based on the granularity of basic blocks needs to be reconsidered, because it would dramatically increase the overall communication load on system buses.

2.4 The Proposed Object-Based Codesign Approach

To address the aforementioned problems, we have proposed a new codesign approach, which is object-based and oriented towards the distributed target architecture with layered bus structure for communications among system hardware/software components. Preliminary work was reported in [LJC95]. Figure 2.6 illustrates its design flow.

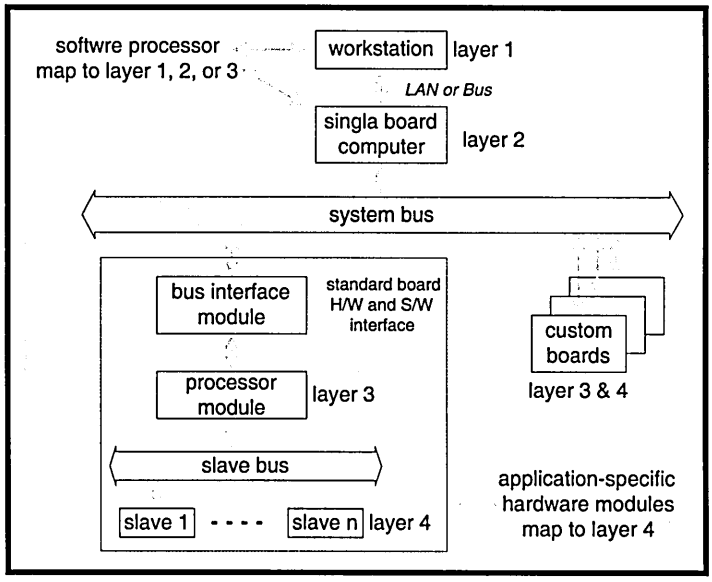


Figure 2.5 Target Architecture with Layered Bus Structure

In our approach, the codesign system modelling technique is largely built upon the PARSE methodology. The *PARSE process graph* has been employed to describe process structures and their precise interactions. This specification is biased on neither hardware nor software implementations. To specify the detailed behaviour of hardware/software components and their communications, the **C**odesign **B**ehaviour **S**pecification **L**anguage (Co-BSL) was designed specially in our project. Guidelines for the conversion from Co-BSL program into intermediate-level presentations (C, VHDL, and the communication configuration on the distributed target architecture) are also provided. The functionality of the codesign system can thus be verified early in the system-modelling phase by simulations in VHDL programs (stage 1).

In hardware/software partitioning phase (stage 2), the profiling information acquired from the VHDL simulation for the verification of system functionality is used to identify the time critical regions and the communication-intensified channels. It facilitates the dispatch and allocation of hardware/software components and communication channels in stage 2.

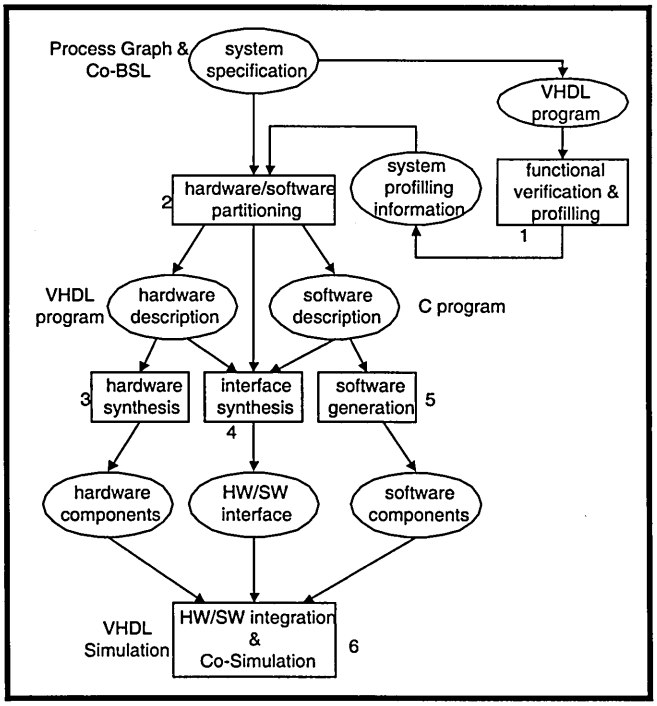


Figure 2.6 Proposed Codesign Framework

Hardware partitions created from the partitioning phase are converted into VHDL programs (stage 3) whereas software partitions into C programs (stage 5). While the performance of hardware component is evaluated in the hardware-scheduling algorithm, namely *List Scheduling Algorithm*, the performance of software component is evaluated in simulations of ARM SDT tool-kit. The individual performance obtained from the evaluation above is then annotated in a unified VHDL program that is simulated in a VHDL simulation environment for system performance evaluation.

The novelty of this part of the work is that the *virtual prototyping* technique [PLCV97] (also see Chapter 5) is applied to the distributed system architecture. In order to support this application, a layered bus communication structure was designed first, together with an asynchronous bus protocol. They form a platform used for prototyping system target architectures virtually in VHDL programs that are supported by the VHDL packages and libraries. Second the interface between hardware and software

components can accordingly be synthesized together with the allocation of communication channels in this virtual environment (stage 4). Third, the system performance can be evaluated in VHDL simulations (stage 6) instead of the simulation in physical prototyping.

This technique is flexible and has a relatively low cost because no special hardware equipment or component are needed. Besides, the codesign production time can be significantly reduced because of the following three advantages:

- Implementation of hardware components in a VHDL program is supported by the existing commercial hardware synthesis tools.
- C compilers and assemblers readily implement the software components in C program.
- The codesign system itself can be developed concurrently with the fabrication of hardware components due to the support from VHDL simulation environment.

Details related to each phase of the proposed codesign approach will be discussed in the next two chapters, and the contributions from this thesis will be mentioned where appropriate.

Chapter 3

System Modelling and Functional Verification

In this chapter, a review of the related research is first given, which supports our proposal of the object-based codesign approach. Next, the modelling technique and the specification notations employed in our approach are introduced. Guidelines for the conversion from high-level co-specification down to intermediate-level descriptions are also presented. Finally, the system functional verification and system profiling in VHDL simulation with token passing protocol are explained.

3.1 Modelling and Specification for Codesign System

A previous investigation [CLJ95b] has shown that various system models and specification tools have been adopted by the codesign approaches surveyed in Chapter 2. The system means involved are:

- CSP
- Petri nets
- Codesign Finite State Machine
- Program-State Machine
- Parallel Random Access-Machine

The system specification means include:

- C or C-like, C++, and Esterel
- SpecCharts
- VHDL, Verilog, and HardwareC
- SpeedCHART, Occam II, and Petri nets
- Statemate^{TM}, SDL, and SDF graph
- OOFS

An analysis [CLJ96b] of these models and specification styles has revealed an obvious drawback, in which most of specification techniques are either *hardware biased* or *software biased*. It has also indicated that the codesign techniques are historically inherited from the *high-level hardware synthesis*. These factors play an important role

on both initial specification and subsequent development processes. Whilst it is appropriate for certain applications in which software components only perform a minor role, it does not allow for the migration in functionality from hardware to software required by changes in application domain. In addition, a tendency in the design of embedded system is towards a significant reduction of design time by re-using previous designs and exploitation of available components. Existing codesign methodologies provide rather limited support in component re-use.

Furthermore, the specification in the early stage of codesign development should provide a good balance between the abstraction power and the ease of implementation. As the complexity of codesign system is dramatically increasing and the codesign technique is maturing, the latest development in codesign methodology demonstrates a strong desire to seek more powerful system specification abstractions. Our analysis has led to the hypothesis that the object-oriented analysis/design technique can help harness the complexity and at the same time produce more reliable and reusable codesign system. Since there are no standard and widely accepted system models and notations in codesign, the object-oriented heterogeneous specification seems to meet this need.

3.2 Object-Orientation in Codesign

An object-based analysis and design technique has the following potential benefits [Gra94]:

- Well-designed objects in object-based systems are the basis for systems to be assembled largely from reusable modules, leading to higher productivity.
- Reusing existing classes which have been tested in the field on earlier projects, leads to higher quality.
- The message passing paradigm allows for a much better, overall description of the system.
- Encapsulated object types promote the implementation-neutral system descriptions and facilitate subsequent object refinement in different designs.
- Object-based systems scale up better from small systems to large systems.

The benefit of modelling a codesign system in the object-oriented concept is considerable. Firstly, safe software development requires that design notations capture

the structural and behavioural properties of the design, i.e., the identification of concurrent process objects and their modes of interaction. This allows the dynamic behaviour of the interacting components to be verified at an early stage of the design cycle. Secondly, object-based design approaches provide object decomposition and encapsulation. The decomposition of a problem into appropriate objects and communication path permits the application of composition rules that ensures the logical correctness of the design [GM93]. Object encapsulation allows the isolation of sub-systems and separate development within the appropriate hardware or software design notation.

Previous research into object-based analysis and design for codesign has already made some progress though it is still in an early stage. Nam S. Woo has been working on the co-specification method for codesign [WDW94]. By using Object-Oriented Functional Specifications (OOFS) a system is divided into three groups: hardware, software, and codesign, which are then treated separately. Although OOFS for codesign group can been translated into C++ and Bestmap-C for the implementation of software and hardware respectively by the compilers, the estimation and evaluation of system performance have yet to be worked out. John Forrest has focused on heterogeneous specification and implementation-independent descriptions for codesign systems [For95]. The basic concept in his work is that a system is described as a set of concurrent modules, each module has a number of ports, and the associated module ports are connected. It reflects the common step of hierarchical decomposition. Two sets of notations, unbiased to hardware or software, have been proposed: an outline one and a reflection of part of it via C++. However, how the transition from these notations to the low-level implementation is smoothly carried out (*cosynthesis*) and how the estimation and evaluation are integrated into this approach (*coestimation and coevaluation*) remain unknown.

3.3 The Co-PARSE Object-Based System Modelling and Specification Method

In order to address the problems in existing codesign approaches highlighted above, a new object-based codesign methodology has been proposed in this project. The system modelling technique adopted in it is based on the PARSE approach [GGJ95] [GJG93], in which its modelling technique and specification means are well shown. Our codesign

approach differs from previous object-oriented work undertaken in codesign discipline, in which the previous work focused upon the use of object-oriented analysis techniques whereas this new approach focuses upon the benefits of object-based codesign such as component re-use.

3.3.1 PARSE Modelling Technique for Codesign

The principles of managing complexity in a system are listed as follows [CY91]:

- Abstraction
- Encapsulation
- Inheritance
- Association
- Communication with messages
- Pervading methods of organization
- Scale
- Categories of behaviour

Various analysis and design methods incorporate some of these principles. Object-orientation in software engineering can be simply characterized by two key features that are glorified by *encapsulation* and *inheritance*.

PARSE supports an object-based approach to parallel system development, utilizing a hierarchical decomposition and refinement design style. It can be used together with recognized object-oriented analysis techniques [Ala90]. Originally it was developed to support the production of robust, reusable parallel software. Current research has focused on further exploiting the usage of its process objects for the development of distributed software systems based on the "client-server behaviour" model and producing CASE support in an industrial setting [RPJ+96] [LG96] [HG97].

The PARSE approach maintains two important properties that are encapsulation and partial inheritance [Sad95]. It provides the designer with a high level abstraction of system definition, which is unbiased by the system target architecture and programming language. Because of this, its use in codesign has been proposed [LJC95].

PARSE provides high-level implementation-neutral abstractions for the specification and supports hierarchical decomposition, encapsulation and component re-use. A summary of the usage and operation of PARSE can be found in [GGJ95]. The initial PARSE specification, can then be further refined ready for partitioning. The partitions consist of objects designed for hardware synthesis, and objects destined for software compilation. The process of translating PARSE into a range of software design notations for software synthesis has been well understood [RSJ95] [RPJL+96] and in this project the work has been done for translating the objects earmarked for hardware implementation into VHDL equivalent components [CLJ96a]. In addition, the user can simulate a process graph description in the early stage of system development only by converting communication paths and path constructors into their VHDL counterparts and providing interface information between the processes.

Table 3.1.

	State Transitions	Behavioural Hierarchy	Concurrency	Program Constructs	Exceptions	Behavioural Completion
PARSE	○	●	●	●	○	●

● Feature fully supported, ○ Feature not supported

We can summarize the discussion above in the following five points:

1. From the viewpoint of key features supporting the development of embedded systems [Gaj94], Table 3.1. [CLJ96b] illustrates the important characteristics PARSE notations possess, which indicates PARSE can support most features needed for the specification of codesign systems.
2. PARSE approach is object-based and PARSE notation is not hardware/software prejudiced, which corresponds with the requirement of specifying a hardware/software unbiased codesign system in object-oriented heterogeneous specification means. In addition to PARSE notations, the fundamental principle behind PARSE such as modularity, adaptability, reusability and maintainability is also highly adaptive. It is appropriate to say that not only have PARSE notations been employed, but also its strategies managing complexity.
3. PARSE process object notations help the designer to produce reusable modules by encapsulation. It is necessary to be highly cautious about the claim of reusability because there are two fundamentally different ideas of reusability. One is to make use of modules from project to project and the other is reusable in the same project. Although inheritance is partly responsible for re-use in the same project, it can

compromise this objective. The reason for this is that inheritance sometimes exposes implementation details to an object's clients [Gra94]. The strength of the PARSE approach is, however, mainly from the encapsulation and abstraction compared with its support in inheritance. This can benefit the reusability from project to project by exploitation of the encapsulated module developed in the previous projects as the off-the-shelf components.

4. PARSE is capable of abstraction of concurrency. Although it is a general feature for HDLs to support the concurrency, its level is relatively low and the concurrency is basically embodied in combinational circuits rather than the system itself. PARSE offers a rich concurrency abstraction at the system level.
5. PARSE is a mature approach involving several years work at the collaborating institutions. PARSE CASE tools are being developed, which could benefit codesign projects. In addition, the codesign approach plus its supporting tools or development environment based on PARSE methodology will not run into problems as copyright of software tools and other legal issues.

3.3.2 PARSE Notations

Three representational formalisms, that is process graph notations, textual BSL, and a schema of relational tables, support the representation of PARSE design. It is possible to translate a PARSE design expressed in one formalism to an equivalent design expressed in another [Gray95]. The proposed codesign methodology has employed process graph notations as a high-level specification means. A hierarchical structure of PARSE process graph notations is illustrated in Figure 3.1 and the PARSE process graph notations are summarized in Figure 3.2 [GGJ95].

PARSE views a system as a collection of concurrent, hierarchically structured, interconnected components known as process objects (Figure 3.2). The designer must categorize each process object either as function server (ellipse icon), data server (rectangle icon) or control process (round-angle icon).

Function server objects are passive and encapsulate some well-defined function or behaviour. Data server objects are also passive and encapsulate important shared data structures in a system. Control processes are active objects, which coordinate other objects in a system in order to achieve the application's goal. Every process object

inherits some predefined behavioural properties from its general class. These process objects might show internal concurrency and consequently can be hierarchically decomposed into lower-level process objects. This classification supports the expression of common design heuristics and abstractions, and enhances the amount of design detail captured at each level. It permits the developer to express the system design in a manner that is compatible with further refinement in both the hardware and software arenas.

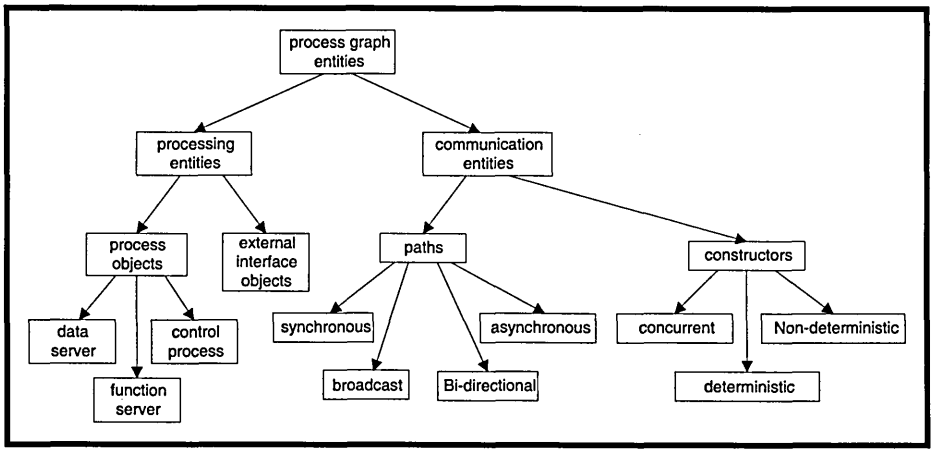


Figure 3.1 Classification of Process Graph Entities

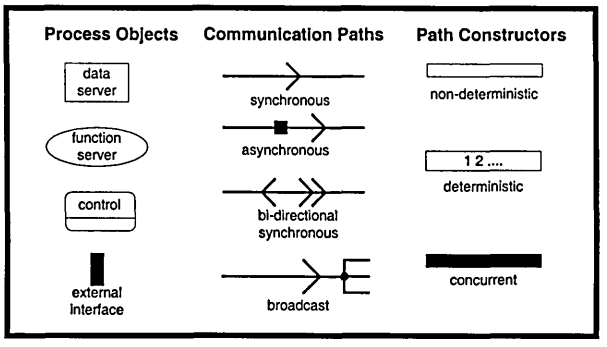


Figure 3.2 PARSE Process Graph Notations

At the stage of logical, architecture-independent design, process objects may only communicate by exchanging typed messages along communication paths; no directly shared data space is permitted. This design approach enforces the encapsulation of important and useful data and functionality in the system at hand. It further produces an architecture-independent design that can be mapped onto specific message-passing or shared memory machine architectures at a later stage in system development. Four classes of communication paths are provided: synchronous, asynchronous, broadcast and bi-directional synchronous, the latter modelling a coupled request-reply message exchange.

PARSE provides path constructors, which allow the designer to represent different modes of handling incoming messages in the situation where a process object possesses multiple incoming communication paths. These modes include: concurrent input handling (implying a further level of decomposition into lower parallel process objects), non deterministic selection and deterministic (prioritized) selection, and the default case in which the sequential ordering of message receipt is defined in the full internal description of the process object.

In the original PARSE approach, in order to support a more detailed description of the behaviour of a parallel system, the process graph entities are represented textually by a ***Behavior Specification Language*** (BSL) [GGJ95] [GB94]. However, since the BSL was initially designed for the development of parallel software it does not take into account the expression of various system constraints and hardware-related mechanics in codesign system. For example, the operations such as binary bit "*or*" and "*and*" operations and binary bit shift operation that are commonlly used in logic circuitry are not included in BSL. The BSL is obviously unsuitable as a high level language for the designer to describe a codesign system and the sequential behaviour of each primitive (i.e. non-decomposable) process object. In particular, it is impossible to describe some of the sequential behaviour in a primitive process object, which is destined for hardware implementation. In our Co-PARSE project, a new language, namely **Co**design **B**ehaviour **S**pecification **L**anguage (**Co-BSL**) has been designed to serve the purpose of codesign specification.

3.4 The Co-BSL Language

While this section is not intended as a textbook to introduce a new programming language, it begins with an overview of the Co-BSL language. Major language features such as program structure, data types, variables, operators, constructs, and communication between processes, are all outlined. The emphasis is laid on the comparison with original BSL language. Examples are given to illustrate the usage of the language.

3.4.1 Overview of the Language

The Co-BSL is primarily designed as an alternative specification means for the proposed object-based codesign approach. It complements the process graph notations to capture the dynamic behaviour of primitive process objects, destined for software and hardware components. Compared with the original BSL definition, the main features of Co-BSL can be drawn as follows:

1. All Co-BSL elements are user accessible i.e. it does not separate the textual portion from the user accessible ones as restricted in BSL [Bra94].
2. Its program portions are convertible to VHDL and C programs rather than the Occam counterparts.
3. Its data type and expression are enhanced to include those especially for hardware components.
4. A new communication channel, by the name of WIRE, is introduced to model the behaviour peculiar to communications between hardware components.
5. A time indication can be created and attached to primitives, which facilitates the cosynthesis of hardware/software with performance constraints.
6. Statements asserting system constraints particularly to hardware aspects are enhanced.
7. Both procedures and functions are included.
8. Object-based features are preserved and those irrelevant to codesign are dropped.
9. Conventional delimiters are used to make the program more readable.
10. Other minor changes are made, which will be mentioned where appropriate.

The BSL explicitly separates the textual representation from the user accessible elements (*primitives*). This artificial division was due to the initial intention to convert the BSL program into a Occam program automatically by a compiler that was then under development. Many of the structures in process graph notation however have very few equivalence in the Occam. This leads to the introduction of the non-automatically convertible part (*the textual representation*) and the automatically convertible part (*the user accessible elements*) in BSL language. This separation is conceptually awkward as a programming language and difficult for user to understand the implementation details. The aforementioned restriction has been relaxed in Co-BSL, although this does not mean that it is impossible for Co-BSL to be automated. Unlike

BSL, Co-BSL does not have the characteristic of user accessible and inaccessible parts, which could otherwise introduce greater complexity in terms of understanding the language itself.

It should be pointed out that like BSL, the Co-BSL adopts some of the important principles in object-oriented programming languages such C++ and Java. Those object-oriented facility provided in Co-BSL are: *abstraction*, *encapsulation*, *message passing*, and *scale*. However, Co-BSL has its limitation, in which the *polymorphism* and *inheritance* that are equally important issues in object-orientation are not supported.

As this research is to establish an experimental codesign approach with the consistent support needed in codesign process, the automatic compilation of Co-BSL program into VHDL, C and communication configuration is not the focus of this project. We instead aim at furnishing Co-BSL with a syntactically verified grammar and conversion semantics that is the topic of the next subsection. To make the context concise, the syntax definition of Co-BSL in the ***bison*** [LMB92] is listed Appendix A for reference. They have been thoroughly checked through GUN software [LO96]: ***flex*** and ***bison***. They are the equivalent Windows versions of LEX and YACC [Joh80] [Ben90]. Due to the lack of space, the Co-BSL grammar checker in the form of C program (the output from the ***bison***) is not included in this thesis but is available on request.

From the codesign viewpoint, the most important element in Co-BSL is the primitive. In the partitioning phase, the primitive is the basic unit to be dispatched to hardware and software components. The primitives assigned to the hardware synthesis are converted to the VHDL description whereas those destined for software implementations are converted to the C program instead. The advantages of adopting primitives as the smallest partitioning units will be discussed in Chapter 4. Other elements, particularly those that are communication-related, are used to synthesize the interfaces between hardware and software components. The communication channels are transformed into a configuration in line with the target architecture with the layered bus target architecture, which is built upon an asynchronous bus communication protocol and other bus communication components we designed in this project. They are the main topics in Chapter 5.

3.4.2 Co-BSL Data Types, Variables and Operators

data types comprise primary and composite types. A primary type is:
INT | REAL | BYTE | BOOLEAN | BIT | OCTAL | HEX | CHAR | TIME
A composite type is either ARRAY or RECORD.

Co-BSL is a strong-typed programming language. Obviously, some of primary types, such as BIT, TIME, OCTAL and HEX, are essential for the user to describe hardware's data property. Besides, although the composite type RECORD is not object-oriented, it needs to be included to represent tokens (discussed in Section5.5.3), which is required by the token passing protocol [Sch92] [Rao92]. The protocol will be discussed later when the conversion of Co-BSL program is dealt with.

variables must be declared before use. An example is as follows:
VARIABLE
num_operation: INT;
delay_operation: TIME;
state_operation: ARRAY [15] BOOLEAN;

operators are classified as:
relational operators: < | =< | > | => | = | <>
shift operators: SLL | SRL | SLA | SRA | ROL | ROR
arithmetic operators: + | - | * | / | REM | MOD
arithmetic (unary) operators: + | -
logical (binary) operators: AND | OR | NAND | NOR | XOR
logical (unary) operators: NOT
concatenation operators: &
Binary and shift operators are essential to specify the hardware's behaviour and therefore included. Following the convention in both C and VHDL, logical operators are applicable to both BIT and BOOLEAN data types.

3.4.3 Co-BSL Control Constructs

Like other ordinary procedural languages, Co-BSL is equipped with a full range of constructs of conditionals and loops, for example

condition

IF THEN or IF THEN ELSE or CASE

loop

WHILE and FOR repetitions

3.4.4 Co-BSL Program Structure

An instance of Co-BSL program is illustrated as follows (not all elements have to be presented though).

```
CODESIGN codesign_name
    constants
    paths
    primitives
    classes
    externals
    executions
    connections
END_CODESIGN
```

The language-reserved words, such as CODESIGN and END_CODESIGN above, are written in uppercase totally. Comments start in two dashes --, which continues until the end of the line. Co-BSL allows user to use all elements listed above in a Co-BSL program though particular description power for codesign system is emphasized within the primitives. Other language mechanics are described below.

codesign_name is an user-defined name representing a codesign system at the top-level.

constants are optional and user-defined as global parameters. An example is:

```
CONSTANT
time-of-duration = 100 ;
```

paths are optional and used to define the communication channels with the type and protocol.

A communication channel can be defined below:

path_name path_type message_type

The path_name is only an identifier representing this path. The path_type is one of the following: SYNC | ASYN | BROD | BIDI | WIRE

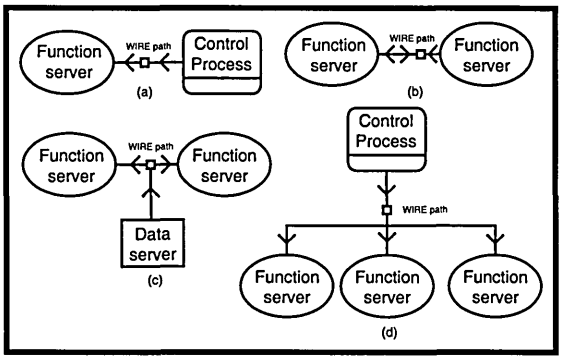
message_type can be any primary or composite data type, which defines the data passing through the communication path. The following example opens three communication channels: in-vector (synchronous), out-data (asynchronous), and status (wire):

```
PATH
in-vector SYNC ARRAY [15] INT;
out-data  ASYN BYTE;
status    WIRE BIT;
```

In addition to those new data types and control structures, Co-BSL extends BSL with a new path type, *WIRE* path. Introducing this type of path is important because it supports modelling behaviours commonly occurring in communications between hardware components. This is inspired by the object class *signal* in VHDL, which is a descriptive abstraction of a hardware wire. The WIRE path serves both to hold changing data values and connect components. A WIRE path is a communication path connecting one component to another. Along this path, the data flows. Connected to the WIRE path, neither the sender blocks itself when the receiver is not ready nor the receiver waits when the new data has not yet arrived. Also, the sent message is not preserved in a queue (*length* > *1*). Furthermore, a WIRE path can be connected to more than two components and there can be more than two potential senders of the path, in which situation a resolution of controversially simultaneous messages sent to the same channel by different path senders has to be involved. This communication semantics is obviously different from previous ones. Accordingly the process graph notations of PARSE have to be extended by adding this new communication path.

Figure 3.3 illustrates some examples of WIRE path. In (a), the control process is the sender of the path and the function server is the sole receiver. In (b), the left function server is both the sender and receiver while the right one is only a sender. The example (d) seems similar as the broadcast communication channel but there is no buffer for any of the function servers. The necessity of this type of path is not only in relation to

modelling the behaviour of hardware component but also to many system-modelling problems. It will be demonstrated in the following examples. The first one is the *Radio Data System* [MW90] [RDS98], the data transmitting component periodically relays information in frames. It does not care whether the receiver is ready and also the data previously sent out is not reserved in a queue for listener's retrieval. Obviously, this kind of connection between the transmitter and the receiver is WIRE type.



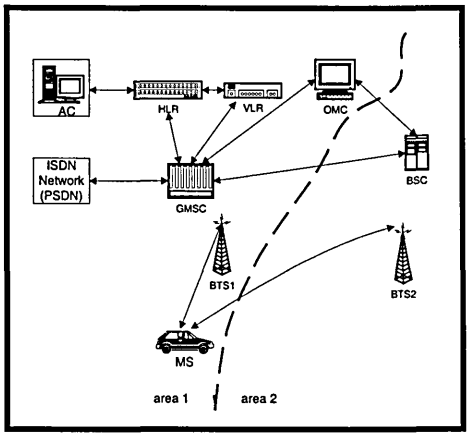


Figure 3.3 Examples of WIRE Path Figure 3.4 Scenario of Handover

The second example is extracted from the GSM mobile communication system [Red95] [MP92]. Modelling the system behaviour concerning the handover procedure of a mobile unit has to employ the path type WIRE. The general scenario of the handover is depicted in Figure 3.4. When a *Mobile Station* (MS) goes across the boundary between area 1 and area 2, the handover occurs. The MS has continuously to monitor the neighboring (area 1 and 2 in this case) cell's perceived power levels and receiving qualities. The MS sends the measurement report back to the currently serving base station to facilitate the decision on when the handover constraints and thresholds are met. The connections between MS and BTSs (*Base Transceiver Stations*) are WIRE type as the MS only monitors the signal's power level and the quality from BTSs.

A third example concerns a computer CPU system. The clock-generating component sends clock signals to various other system components. The connection between the clock generator and other components is again WIRE type.

Because of the similarities between the wire path of Co-BSL and the signal in VHDL, the Co-BSL's wire path are defined in similar syntax and semantics as the signal in VHDL. For example, the wire path assignment is in the mnemonic "= = >" or "< = =", a

process can suspend on a wire path by executing wait statement, and the assignment can be dealyed by the explicite form "AFTER time-expression". Furthermore, Co-BSL is assumed to contain some of wire-path related attributes predefined in the laguage, such as 'Active, 'Event, 'Last_Value, 'Last_Event, and 'Last_Active, which provide information about the wire path.

externals are optional. They are used to declare all external objects in a codesign which have the same properties as primitive objects, but only their interfaces are visible.

executions define all the instances of concurrently active processes that could be primitive sequential processes, external objects, or instantiated classes.

connections instantiate the connections between processes.

primitives are processes that are optional and defined for a collection of primitives. They are declared globally and used to describe the sequential behaviour in the non-decomposable processes. A single primitive looks like follows:

```
PROCESS process_name OF class_type time_indication
    inports
    outports
    constructors
    variables
    function_declaration
    procedure_declaration
    main_sequence
END_PROCESS
```

In BSL, only one timed communication protocol is provided for specifying the time limit to wait for a message from the other end of the communication link. In codesign, however, the maximum time latency in executing a sequential primitive process may need to be specified, which will be used as a system performance constraint during hardware/software cosynthesis stage. In Co-BSL we have introduced the

time_indication. It can be used to verify whether a hardware/software implementation meets the time constraint later in the partitioning and performance evaluation phases.

The main_sequence above is a group of statement parenthesized with BEGIN and END. The statements are delimited by ';'. A single statement could be any one of the following:

/* empty */
| WAIT | WAIT ON NAME | WAIT UNTIL expression
| WAIT FOR expression
| BREAK | CONTINUE | RETURN | SKIP | STOP
| assignment | signal_assignment
| condition_statement | case_statement
| loop | proc_call | io_operation

Note that the wait statements above are important. It is used for activation or suspension of an active sequential process. This is a distinctive feature, compared with the BSL, in which an active process can only suspend at a communicating rendezvous.

Example 3.1 - SR Flip-Flop

The behaviour of a SR flip-flop is described in a Co-BSL primitive (Figure 3.5).

```
PROCESS SRFF OF FUNCTION_SERVER
INPORTS
      S WIRE BIT;
      R WIRE BIT;
OUTPORTS
      Q     WIRE BIT;
      Qbar WIRE BIT;
VARIABLES
      Last_State: BIT := '0';
BEGIN
  IF S = '0' AND R = '0' THEN
    Last_State := Last_State;
  ELSE_IF
     S = '0' AND R = '1' THEN
     Last_Sate := '0';
  ELSE
     Last_State := '1';
  END_IF;
  Q <= = Last_State AFTER 2 ns;
  WAIT ON R, S;
END
END_PROCESS
```

Figure 3.5 Behaviour of a SR Flip Flop

classes are optional and based on the three general system supplied classes. It represents a collection of classes, which could be instantiated (*reused*) in the same project or other projects. A single class looks like as below:

D_CLASS class_name OF class_type

inports

outports

constructors

paths

executions

connections

portconncets

END_CLASS

It includes a number of *inports/outports*, which describe the input and output connections for the class. Constructors are the property of user defined classes and they can be applied to any user-defined class of process objects. A process object may have many *constructors* attached to it and each of them must be labeled with an identifier. The *paths* section defines internal paths between processes. The *executions* and *connections* are used to instantiate the non-decomposable processes and their connections. The final section, *portconncets*, defines connections between internal processes and the formal parameters.

Example 3.2 - GSM Mobile Communication System

This example presents the usage of Co-BSL for specifying a codesign system, particularly focused on the Co-BSL program structure. Detailed sequential behaviour inside primitive is not included because they are not essential here.

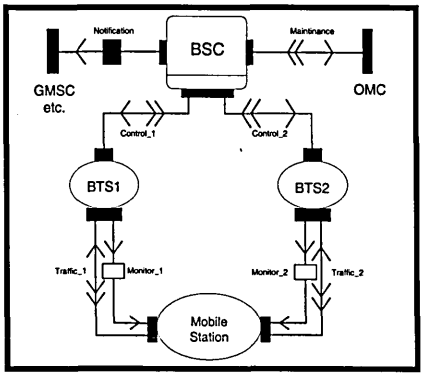


Figure 3.6 Abstraction of Handover

Figure 3.6 illustrates the scenario for handover process in the GSM mobile communication system. Assume that the handover happens within the administrating

area of a BSC and it handles the operation without consulting the GMSC. This BSC then becomes the control process. OMC and GMSC plus other entities are thus external objects. The mobile station has two communication channels linked to the BSC, one for ordinary traffic load and the other for monitoring purpose in a special channel. The mobile station can be further refined. The abstraction of this is shown in process graphs (Figure 3.6 & 3.7) and the Co-BSL program is shown in Appendix C to keep the context concise.

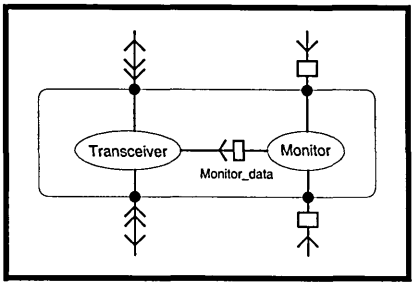


Figure 3.7 The Refined Mobile Station

3.5 Conversions from Co-BSL to VHDL and C

Our previous work has established the conversion rules from BSL primitives and communication channels to VHDL processes and signals [LJC95]. The implementation details and the relevant VHDL packages specially developed in support of the template conversion have been reported in [CLJ96a]. This section is, however, focused on the conversion from Co-BSL design to VHDL/C programs, emphasizing the object-orientation and component reuse. The conversion guidelines described below are aimed at a broader scale. It allows all Co-BSL components to be converted to their VHDL counterparts and preserves object-based features within the Co-BSL design. These features are as follows:

- abstraction
- encapsulation
- communication with messages
- scale
- reuse

Referred to Figure 3.8, the conversion from Co-BSL to VHDL and C has a threefold purpose. First of all, the Co-BSL design needs to be converted into an executable form to check the system's correctness in terms of its functionality. This allows the dynamic

behaviour of the interacting components to be verified at an early stage of the design process. Second, this execution can provide the essential profiling information to facilitate the hardware/software partitioning process. Third, after partitioning phase, primitives in a Co-BSL design are dispatched to hardware and software components that will eventually be allocated to a system target architecture, i.e. *cosynthesis* as named in some literature. Conversions bridge the gap between the system-level specification and the low-level implementation.

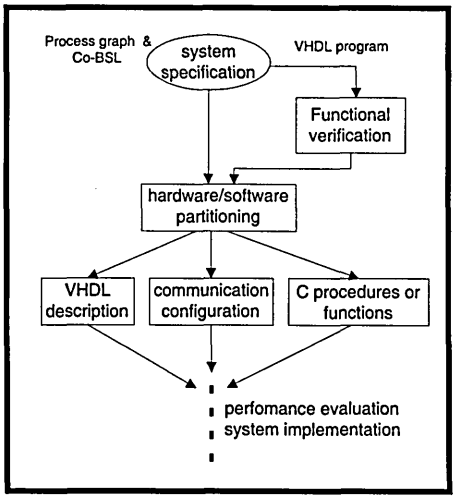


Figure 3.8 Conversion from Co-BSL to VHDL and C

The reason for converting the Co-BSL program to VHDL program rather than C program as an executable intermediate language for functional verification in this project is as follows. The conversion from a Co-BSL presentation to its equivalent C description can only take place at sequential process level, i.e. primitive process level in Co-BSL. However, a Co-BSL design can be readily converted into its VHDL counterpart at any level. VHDL models a discrete system as a collection of processes concurrently executing and passing messages through *signals* [LSU89], so does the modelling in Co-BSL. In addition, a VHDL *process* can be viewed as a portion of procedural language program such as a *function* in C [PB96], which indicates that a portion of C program can been viewed as a portion of VHDL but not the vice versa. The following content, therefore, lays stress on the conversion from Co-BSL presentation to its VHDL counterpart. The conversion into C representation will be mentioned wherever appropriate. The discussion on cosynthesis of the interface between hardware/software components will be delayed until Chapter 5, where it is described in conjunction with the virtual prototyping technique.

The conversion of a Co-BSL design to their VHDL counterparts is established by mapping the Co-BSL design constructs to the corresponding VHDL descriptions. A Co-BSL design models a concurrent system as a set of communicating sequential processes and a VHDL design can also be modeled as a set of communicating sequential processes, an essential feature for the succinct description of both the macro-parallelism and the micro-parallelism present in digital hardware [JDV92]. This property indicates that the conversion can maintain the original design structure.

A Co-BSL program can be represented as follows:

CODESIGN codesign_name

- constants
- paths
- primitives
- classes
- externals
- executions
- connections

END_CODESIGN

The conversion is tackled in accordance with the main elements in the Co-BSL program shown above.

3.5.1 Co-BSL Program

A Co-BSL program can be converted into a VHDL entity without port declaration and the architecture of the entity is created with the component instantiations in accordance with the executions in the Co-BSL program, which will be explained where appropriate.

3.5.2 Constants

Constants can be declared in a VHDL package ready to be used globally.

3.5.3 Paths

The paths are in fact internal signals declared in the architecture body of the entity, which is the VHDL counterpart of the Co-BSL program.

3.5.4 Primitives

The primitives are a collection of Co-BSL primitive and only the conversion for single primitive needs to be identified. A single primitive can be converted into a VHDL design entity, consisting of *entity declaration* and *architecture body*. The information in its inports/outports is transferred into the *entity port* and others into the *architecture body*. In particular, the sequential behavioural description in Co-BSL is readily mapped to a VHDL process. Although the VHDL description does not provide the same explicit syntactical difference as exist in Co-BSL primitives (*control process*, *function server*, and *data server*), the contexts of the VHDL program clarify the semantic difference. These VHDL entities are stored in a library for reuse (*instantiation*). The similar principle can be applied to the conversion into C program but a Co-BSL primitive is directly converted into a C procedure or function.

The collection of Co-BSL control flow constructs is in fact a subset of VHDL's, which control the execution flow in a primitive. The Co-BSL control flow constructs can therefore be mapped directly to equivalent control flow constructs in VHDL. Because the set of Co-BSL control flow constructs is designed as an intersection of both VHDL and C control flow constructs, the Co-BSL control flow constructs can also find their equivalence of C control flow constructs. The correspondence is so straightforward that they are not discussed further here. Contrary to control flow constructs, the implementation of the communication between primitives is a complex task, which is detailed as below.

```
type Handshake is (Inactive_sink, Active_sink, Inactive_source, Active_source);
type Token is
  record
    Status: Handshake;
    Color: Color_type;
end record;
```

Figure 3.9 Token Type Definition

3.5.5 Communication Channels

A handshaking protocol [Sch92][Rao90] has been adopted for synchronization and manipulation of Co-BSL communication channels. The protocol named "*Token Passing*" plays an important role in it. Fundamental to this protocol is the definition of an enumeration type *Handshake*, a record type Token and its resolution function which

is called *Bus Resolution Function* (BRF), because of its simulation of bus functionality in a digital system. Figure 3.9 shows the VHDL type definition for Handshake and Token. Color_type is a user-defined record type containing any signal types required by the particular model. To keep the content concise, the Bus Resolution Function has been put into Appendix B.

The function of a handshake protocol is described as follows.

1. Initially, the output assigns inactive_source to the status field of the signal while the input assigns inactive_sink to the status of the signal. The bus resolution function makes the value of the signal the same as that coming from the input. Both the input and the output can read this resolved value which specifies, by convention that the output may now place a new token on the signal.
2. When the output has a token to be placed on the signal, the output assigns active_source to the status field of the signal. The input is still assigning inactive_sink to the status field of the signal. The bus resolution function reconciles these differing assignments by placing a copy of the token from the output onto the signal.
3. The input can now see an active_source status field value on the signal. When the input is able, it copies this token and subsequently changes its assignment to the status field of the signal to active_sink. The assignment from the output remains unchanged, active_source. The bus resolution function now places a copy of the token from the input onto the signal.
4. The input can now see an active_source status field value on the signal. When the input is able, it copies this token and subsequently changes its assignment to the status field of the signal to active_sink. The assignment from the output remains unchanged, active_source. The bus resolution function now places a copy of the token from the input onto the signal.
5. When the output sees the active_sink status field value on the signal, it knows that the token it placed on the signal has been accepted by the input. The output then assigns inactive_source to the status field of the signal. The assignment from the input remains active_sink. The bus resolution function now places the token from the output onto the signal.
6. The input sees an inactive_source status field value on the signal and prepares to receive a new token by changing its assignment to the status field of the signal to

inactive_sink. The output's assignment to the status field of the signal remains inactive_source. The resolved value placed on the signal is the token from the input. This handshaking process now loops back to step one.

This protocol embodies the communication style of a synchronous bus in a digital system, which is widely used in simulation for computer communication via various types of synchronous buses. It is synchronized between sender and receiver because the sender has to be blocked until the receiver issues handshake signal "active_sink", which informs the sender that the receiver is ready to read the message from the sender. This protocol can be used in the template conversion of synchronized communication path (synchronous and bi-directional) in Co-BSL whereas new mechanisms have to be sought to support the other two asynchronous communication paths (asynchronous and broadcast). Based on this token passing protocol, the conversions of Co-BSL communication paths to VHDL templates are described as follows.

3.5.5.1 Synchronous Communication

The synchronous communication template invokes two procedures. The sender and receiver are situated in two concurrently active processes that are synchronized upon tokens passing between them. Procedure syn_transmit(signal T: inout Token; variable ColorT: Token; delay: time; wait_delay: time) (Appendix B) is invoked by the sender process and Procedure syn_receive(signal T: inout Token; variable ColorT: out Token; delay: time; wait_delay: time) (Appendix B) by the receiver process.

3.5.5.2 Synchronous Bi-directional Communication

A bi-directional communication template models the client/server interaction between sender and receiver. Both the client and server are blocked at the path before they finish communication. Therefore, a bi-directional communication path can be replaced by two synchronous communication paths. The client is designated as the sender of the first synchronous communication path and the receiver of the second path. In opposition, the server is the receiver of the first path and the sender of the second path. Any of the message types such as *request*, *accept*, or *reply* can be attached to the tokens traveling along these two paths.

3.5.5.3 Asynchronous Communication

A VHDL process running concurrently with the sender and receiver processes has to be created as a communication buffer (illustrated in Figure 3.10). This template maintains a FIFO queue. The sender's priority is higher than the receiver's and the receiver has to be blocked when the queue is empty, but the sender is never blocked because the queue is made up by dynamic memory allocation which allows for a literally infinite queue.

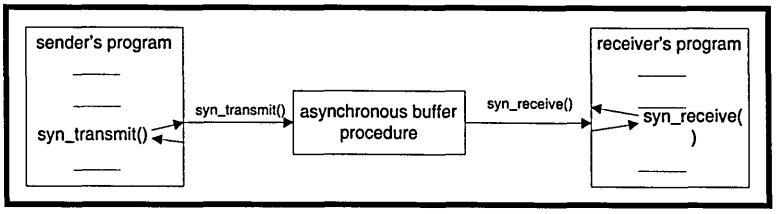


Figure 3.10 Template for Asynchronous Communication

Procedure asyn_buffer(signal T_in: inout Token, signal T_out: inout Token, delay: time) is shown in Appendix B. Procedures in_que, number_of_que and out_que can be found in the package Queue_definition which has been developed separately. Their functions are:

- in_que(que_point, temp), token temp is put into the queue.
- out_que(que_point, ColorT), the first token in the queue is pulled out into ColorT.
- number_of_que(que_point, tmp), the total number of elements in the queue is put into tmp.

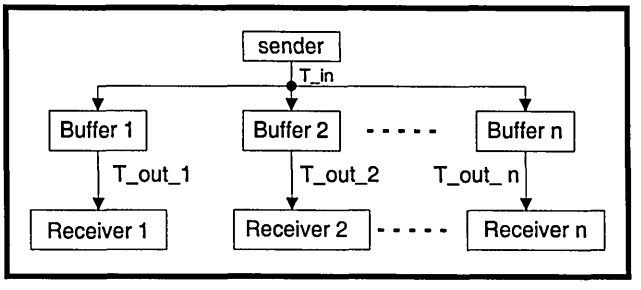


Figure 3.11 Template for Broadcasting Communication

3.5.5.4 Broadcast Communication

Broadcast communication is a "non-blocking send" and a "blocking receive operation". The sender broadcasts messages in a channel without considering if the receivers are ready to receive them and the receiver will be blocked if there is no message in the channel. The receivers listen to the channel and receive the messages in the same order in which the sender sent them out. Because receivers can receive messages at different times, different buffers for receivers are constructed to keep the messages in the queues

in accordance with the different receiving speed. The template for this scenario is shown in Figure 3.11, in which two procedures are invoked:

- Procedure bro_sepera_buf (signal T_in: inout Token, signal T_out: inout Token, que_point: inout queptr; delay: time; wait_delay: time) (Appendix B)
- Procedure bro_syn (signal T: inout Token; wait_delay: time) (Appendix B)

N buffer processes are managed in the template, one for each receiver. Technically, a special "Receiver" has been attached to T_in to participate in the hand-shaking process, representing all buffer processes. Without it the sender would have to finish hand-shaking procedure in turn with every buffer for the same message broadcast in T_in. These buffer processes now all listen to the signal T_in and choose the token with the status of "active_source" sent out by the sender. In this cycle of handshaking it is assumed that the real message is attached to the token.

3.5.5.5 Wire Communication

Since wire communication embodies the communication in *signal* in VHDL, its conversion to VHDL template is consequentially straightforward and the handshake protocol is now no longer needed for synchronization between sender and receiver. The wire path can directly be declared as a signal with the Color_type as a user-defined record type in VHDL. The path sender is a *source* for the signal and connected to an *out* port while the receiver is connected to an *in* port. If both sending and receiving are required in a Co-BSL primitive, an *inout* port can be declared in its counterpart VHDL process. Finally, in case of sending messages simultaneously to the same communication path by different path source, a resolution function has to be provided by the designer in order to resolve the conflicting sources, which is a common practice in VHDL.

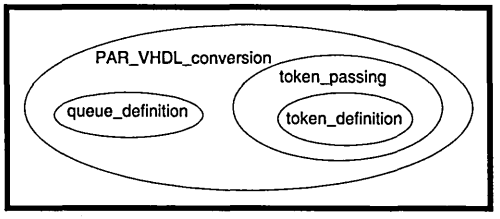


Figure 3.12 Relationship between Packages

In order to facilitate the conversion, we have developed four VHDL packages, integrated into user's VHDL simulation programs. These packages play an important

role in our methodology. As the reusable VHDL components, they help check the correctness of the system specification and provide a test platform. Because of them, dealing with communication channels in Co-BSL program has become so easier that the user only needs to declare a *signal* in VHDL and invokes the relevant procedures from the package *PAR_VHDL_conversion*. This package is designed as the only interface between application VHDL programs and the four packages (see Figure 3.12). The package *queue_definition* provides the operations commonly used in asynchronous communication path and the packages *token_passing* and *token_definition* provide both token's definitions and manipulations.

These packages have been tested by a series of experiments on a test bench [CLJ96a]. Because of the space limit, the VHDL source files of the packages are not listed, but it is available on request. Although an executable recursive algorithm for translating a Co-BSL description into its VHDL counterpart is possible as described in [Gaj94], it is not intended to be the focus of this research.

Finally, as shown in [LJC95], Co-BSL path constructors can be readily mapped to VHDL templates. In addition, the non-deterministic path constructor can also be implemented, supported by a random number generator [Bak93] to handle the communication on more than one path at the same time and treat them in random order.

3.5.6 Classes

Converting Co-BSL classes to its VHDL equivalence provides codesign with the scalebility and reusability. Note that the *class* in Co-BSL is not exactly the same jargon as the abstract data type, class, as in C++ because the data inheritance is not emphasized in Co-BSL that is focused on encapsulation, message passing, scalebility, and reuse. A Co-BSL class is a collection of primitives (or previously defined classes) grouped together, connected in paths so as to be reused as a scaleable collective component. Its assembly mechanism in primitives is rather similar to the assembling of hardware structure embodied in VHDL, which points out that VHDL contains all required elements to support this conversion.

In reality, only rules guiding the conversion from a single Co-BSL class to its VHDL counterpart need to be established. A Co-BSL class can straightforwardly be mapped to a VHDL entity and put into design library for reuse. Its architecture body comprises the *component instantiation statements*, which reuse the components converted from those primitives and stored in the library. The inports and outports are written in the *port* of the VHDL entity. Besides, path constructors can be implemented in the individual instances of concurrently active processes and the paths are declared as internal signals in the component's architecture body. As path connections and portconnects have been integrated into individual executions, they do not manifest themselves in VHDL program explicitly. Since the primitives, external objects, or instances of other class object have already been converted and stored in a library as VHDL component, *executions* in the class are implemented by using VHDL *component instantiation statements*.

3.5.7 Externals

Externals are a group of external objects that have the same properties as primitives but only their interfaces are visible. A Co-BSL external object corresponds to a VHDL *component declaration* and its instantiation can be implemented in *component instantiation statement* in VHDL

3.5.8 Executions & Connections

They were dealt with previously in the conversion of *classes*.

3.5.9 An Example of Conversion of Co-BSL into VHDL

In this example, the Co-BSL program created in example 3.2 has been converted into a VHDL skeleton program to demonstrate the viability of the guidelines set up in this chapter. Due to the space limit in the content, we have put both its Co-BSL program and the converted VHDL source file in Appendix C for comparison. In addition to substantial comment in both programs, further explanation is needed.

In the VHDL program, five Co-BSL primitives have been converted to five VHDL entities: BSC, BTS1, BTS2, Transceiver, and Monitor. They are compiled and installed in the VHDL library *WORK*, in which there are other VHDL components: *component Ext_GMSC* and *component Ext_OMC* that have been converted from the external

objects *GMSC* and *OMC* in its Co-BSL program. In addition, the Co-BSL class: *D_CLASS Mobile_Station* has been converted to the VHDL entity definition: *entity CLASS_Mobile_Station* which instantiated the components: *Transceiver and Monitor.* The Co-BSL program: *CODESIGN Handover* has itself been converted to the VHDL entity: *entity CLASS_Mobile_Station.*

The communication paths are all converted into VHDL signals as indicated in the previous sections. Particularly, the path *Monitor_data* in the Co-BSL class definition has been converted to the internal signal *Monitor_data_wire* in the entity *CLASS_MOBILE_Station.* Other communication paths in the Co-BSL program are *Notification*, *Maintenance*, *Control_1*, *Control_2*, *Traffic_1*, *Traffic_2*, *Monitor_1*, and *Monitor_2*. They are all converted into corresponding VHDL signals in the entity *Handover.*

3.6 Functional Verification in VHDL Simulation

In the proposed Co-PARSE methodology, the correctness of system function and the satisfaction of performance constraints are verified in two stages:

1. System functional verification in VHDL simulations supported by the token-passing protocol
2. System performance evaluation in VHDL co-simulations supported by virtual prototyping technique (detailed in Chapter 5)

The VHDL simulation in this chapter has twofold advantages: functional debug and system profiling. This section is only concerned with the system functional verification. The system profiling will be discussed in Chapter 4 and the system performance evaluation will be dealt with in Chapter 5.

The functional debug in the system development cycle can find design flaws earlier so as to avoid potentially huge cost of a late patching-up that has been a well known principle in Software Engineering [Pre94]. The functional verification adopted in this project relies on the VHDL simulations supported by token-passing protocol. This method requires a Co-BSL design converted into a VHDL program and simulated in a VHDL simulation environment. The token-passing protocol is mainly used for

synchronization and communication between active processes. It has sound theoretical base and successfully been used in a number of applications [SJA93] [Sch92] [KM+97].

In addition to examples shown in this thesis (*Arithmetic Coding System* in Chapter 4 and *Radio Data Computing System* in Chapter 6), two extra case studies (*GSM system and RDS system*) have already been undertaken and published in [CLJ97] and [CLJ98b]. The VHDL packages and the conversion rules established in this chapter have been tested by these case studies in order to validate our methodology at the stage 1 and 2 as shown in Figure 2.6.

A complete example with both functional verification and system profiling will be given in the next chapter. It is therefore not our intention here to examine this issue further.

Chapter 4

Design Space Exploration

The *Design Space Exploration* is a synonym of *Hardware/Software Partitioning* in the codesign society. Because of its complex nature and the lack of systematic investigation into this field, we have conducted an extensive survey of current research into this area. The original result from the survey has been published in [CLJ98a].

In this chapter, the background of design space exploration is first given, followed by a review of state of the art in this research field. The review is established on our previous investigation [CLJ98a]. Next, the profiling technique together with the system functional verification in VHDL simulations is detailed. The benefit of introducing object-orientation in this field is highlighted. Finally, possible improvements to this technique are suggested.

4.1 Background

As mentioned in the introductory chapter, in comparison with the traditional design path, the codesign approach maintains the flexibility of exploring alternatives in the design space and therefore results in the best solutions to an application domain. This design space exploration is also known as *Hardware/Software Partitioning* in the codesign school, where a codesigner or a tool assigns system components (*functionalities*) to hardware or software implementations. In general, the partitioning problem consists of two different types: *homogeneous* and *heterogeneous*. The homogeneous partitioning is solely concerned with dividing a pure software or hardware system into its components. In the case of pure hardware system, the major objective of partitioning is to satisfy various system constraints such as power consumption and circuitry area, whereas the objective of partitioning a software system is typically to increase the component utilization, speedup the system execution, and reduce the overall communication overhead. Although the homogeneous partitioning problem is still an open research topic, it is fairly well established [VG95][Hua85]. The partitioning problem in codesign is a heterogeneous one. Therefore, in this chapter, we will concentrate on the heterogeneous partitioning problem.

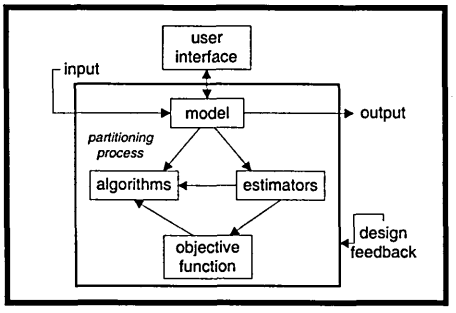
A partitioning technique has to be associated with some kind of attributes with which one can decide the *goodness* of a specific partitioning scheme. These attributes are also called *metrics* in other literature [GVNG94], which could build on monetary cost, execution time, power consumption, circuitry area, memory size, testability, reliability, reusability, and so forth. Individual technique only needs to coordinate part of the issues aforementioned. At present, the metrics overwhelmingly adopted in other codesign methodologies are execution time (*performance*) and monetary cost (*cost*) although there are exceptionable cases such as [JE+94]. We too focus on these two metrics.

The difficulties with heterogeneous partitioning lie in the fact that there are inherent differences in the model of computation used for implementation of hardware and software models, and the two computations proceed at very different rates. Furthermore, the different execution rates cause variations in the communication rate between hardware and software components so as to entail a higher communication overhead due to the necessary handshake and buffering mechanisms [Kum94]. Another fundamental difficulty is that the accurate evaluation of the goodness of a partitioning is utterly based on the attributes that are closely related to the implementation details, whereas there is no implementation at all during partitioning phase. A less accurate technique is therefore created, which is named the *estimation*. The estimation technique enables a codesigner to select the best solutions by weighing the attributes resulted from "*rough implementations*", which is featured as the tolerance of inaccuracy and the high fidelity of the relative goodness of any two partitionings under estimation.

It is obviously inadequate to divide a codesign system solely based on estimation and approximation since there are several numbers of factors that could affect the calculation of attribute. First, the optimization of object code by compiler and the utilization of processor pipeline make the estimation of software timing extremely difficult. Second, the system-level synthesis for hardware is hard to predict in terms of execution time and resource allocation owing to the variable efficiencies of scheduling and allocation algorithms. Finally, the hardware/software communication overhead depends on its executing mechanism, which could result in the deviation of estimated timing up to hundreds of clock cycles. The partitioning process is therefore destined to

be iterative and a preferred partitioning scheme may have to be evaluated after system implementation.

A comprehensive configuration of partitioning system has been drawn in Figure 4.1 [GVNG94]. In this system, the input is first transformed into an internal model that is functionally equivalent. Partitioning algorithms are then applied to this model, which cooperates with estimator and objective function. The design feedback is used to evaluate the impacts on the implemented codesign system from a specific partitioning scheme.



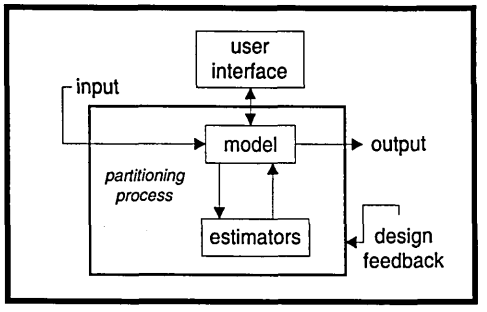


Figure 4.1 Typical Configuration Figure 4.2 The Simplified Configuration

While some of the codesign methodologies abide strictly by this configuration, there are a number of simplified heterogeneous versions of this, in which sophisticate algorithms and objective functions are not pursued and instead the partitioning process relies highly on design feedback and iterative operation. This simplified configuration has been illustrated in Figure 4.2, where only an estimator is retained to facilitate the decision-making and the process is destined to repeat until the design feedback satisfies partitioning objectives.

4.2 Review of Partitioning Techniques

A partitioning technique can be characterized in the following essential issues:

- Input to partitioning
- Partitioning granularity
- Performance estimation
- Performance evaluation
- Target architecture

The input to the partitioning reflects the level at which a partitioning can be carried out. If, for example, a partitioning input is written in VHDL program that could have been

written at three different levels: behaviour, structure, or gate levels, then the partitioning can correspondingly be done at three different abstraction levels. The partitioning granularity is a measure of the size of each partitioned component [AG97]. The partitioning input and granularity are related to each other. For example, the input in a higher abstraction level can only support the partitioning with coarse-granularity. The performance estimation is used to predict the performance over individual partitioning scheme and on the other hand the performance evaluation examines performance by consuming the design feedback from a specific implementation. Finally, the target architecture significantly affects the interactions between hardware and software components, thus the performance estimation and evaluation.

4.2.1 Partitioning Input and Granularity

The partitioning input can be expressed in either program or graphic presentation. The examples listed below have been adopted in the published codesign methodologies:

- C (C++)
- SpecCharts, VHDL or HardwareC
- Occam II
- Signal Flow Graph
- Statemate™ and SDL
- Internal Graphic Presentations

The partitioning input is related to both system abstraction power and partitioning granularity. The system abstraction needs to support partitioning process at a flexible abstraction level. Though hardware/software partitioning can be performed at various abstraction levels, it is mostly performed at the behavioural (*functional*) level. Apparently, all of the partitioning input afore-listed supports behavioural level abstraction thus the partitioning at this level. For a given input, there is only one reasonable partitioning granularity. For example, the input in task-level dataflow graph can only be partitioned into the level composed of tasks (*coarse-granularity*), while the input in arithmetic-level dataflow graph can be partitioned into the level composed of arithmetic-operations (*fine-granularity*).

The partitioning granularity varies from task level to statement block level. They can be roughly divided into two categories: *coarse-granularity* (task, function, and process

level) and *fine-granularity* (statement block level). The partitioning with coarse-granularity is a common practice in the manual partitioning operation. It presumably produces a larger chunk of functionality enclosed in a partition and less communication overhead across the components. In contrast, the partitioning with fine-granularity can usually result in better optimization but often creates too many components and expensive communication costs. A compromise, therefore, has to be made against these merits. As a trial, an experimental work has been reported on dynamically determined granularity for different applications [HE97].

As outlined in Chapter 3, previous research addressed the system specification problem and provided the specification means in an object-oriented fashion but there has been a clear lack of strategy to create an object-oriented (or -based) codesign methodology with consistent support of the object-orientation in the later codesign phases. Here, we would like to argue that from the object-oriented point of view the partitioning input and granularity should be based rather upon objects than upon statement blocks because the benefits from object-orientation obviously outweigh the benefit of optimization resulted from the fine-granular partitioning. The reasons behind that argument can be given as follows:

- The partitions (*objects or object-based entities*) are well designed and assembled largely from reusable modules, leading to higher productivity.
- The message-passing paradigm allows direct mapping of the communication path in the specification into the communication interface in the system target architecture.
- Encapsulated partitions facilitate subsequent object refinement in different design and scale up better from small systems to large systems.
- There is a significant reduction in the number of objects to be partitioned, which makes partitioning algorithms work better and the designers operate easier.

4.2.2 Performance Estimation

The performance estimation supported in the published partitioning techniques can be typified as algorithm, experience, and profiling. In other literature, these terms are remarked as deterministic, statistical, and profiling. The deterministic (*algorithm*) approach requires all data dependencies removed and all costs of components known. It can lead to a very good partitioning scheme, but fails while those elements are

unavailable. The statistical (*experience*) approach is based on analysis of similar system and certain design parameters. The profiling approach is straightforward, which generally yields better results because the partitioning can be determined even when strongly data-dependency exists.

The deterministic approach keeps intervention from codesigner to a minimum. It leads, therefore, to an automated partitioning process, at which the partitioning researches have been aimed. Two representative techniques can be found in [EH92] [GM93]. They both decompose the input into statement blocks and then transform them into internal graphic representations that describe the data dependencies and are used to estimate the system performance and the hardware cost. While they provide better optimizations for the partitioning problem and offer the promise of automation of the partitioning process, two major drawbacks with this technique should not be underestimated:

- Since the input has been decomposed into statement blocks during partitioning phase, the object structure has profoundly been broken apart and the object-orientation is no longer preserved in the design and implementation phases.
- Both techniques can only be applied to relatively small scale problems and the target architecture assumed is simplified into one processor, one hardware component, and single system bus, which neglects the system target architecture in terms of its impacts on system performance.

Another deterministic technique is proposed by [JE+94]. In it, a partitioning problem is formulated as finding a subset of the program regions suitable for hardware implementation and able to gain the greatest system speedup, on condition that they can be fixed into the hardware limit in terms of logic gates. Although it creates a sophisticated algorithm for minimizing the memory interface traffic, the objective of the partitioning is restricted to rather specific problem domain and this technique, therefore, has its limitations from the application's viewpoint. In addition, it also suffers from the same drawbacks explained earlier.

An improved deterministic technique is seen in [VG92] and [GVNG94]. It decomposes the input to one of three levels of granularity: tasks, subroutines, and statement blocks. It also identifies three distinct partitioning problems: mapping system behaviours to

system components (custom or standard processors), mapping variables to memories, and mapping communication channel to buses. However, its estimation model is mainly concentrated on the issues related to hardware synthesis, such as pin number and chip area and there is no indication in how to support the synthesis of hardware/software interface, based on the target architecture with standard processors, ASICs and buses.

Finally, [HW96] has presented a completely different deterministic technique, where a codesign system is generalized as a heterogeneous distributed system. The hardware/software partitioning is then analogized in partitioning different types of processing engines that could be general processors, ASIPs, or ASICs. The performance estimation model is built upon processing engine's computation and communication costs. This deterministic technique tries to minimize these costs and maximize the advantage of each specific processing engine. This work initiates the investigation into the technique transfer from the performance evaluation for general distributed systems to the performance estimation for hardware/software partitioning. While it opens a new dimension of research into this field, it is still in a preliminary stage and some crucial issues, such as communication cost, need to be more reliably counted in its estimation model. In recent years, however, there is a significant interest in exploring the high level model for estimating the communication cost in codesign system [KM98] [HB97], which could provide a supplement for the estimation model enclosing the communication cost.

From the discussions above, it is evident that the performance estimation for codesign system is an open question.

4.2.3 Performance Evaluation

With the design feedback shown in Figure 4.1, the performance evaluation is used to assess the impact on system performance by the "*implemented*" codesign system in line with its attributes set up during system specification phase. As pointed earlier, here we concentrate on two attributes: system execution time and hardware cost. The "*implemented*" system does not necessarily mean a physically realized system with all real components connected together. It could be any of the following: physical implementation, physical prototyping, or virtual prototyping.

The evaluation relying on the physical implementation is less attractive because of its inflexibility and huge cost.

Thanks to the programmable hardware components such as FPGA, the physical prototyping has been a favorite choice particularly in the hardware and electronic engineering society. Due to the programmable component, the hardware components can now be programmed and connected to the processors through system buses. This kind of technology is mature and also precise in terms of performance evaluation. But, on the other hand, it is costly and less flexible. Some times, it is impossible, particularly on the occasion when some of system components are unavailable. In addition, the connection between the programmable hardware component and the processor is in fixed protocol, which is overwhelmingly aligned to the target architecture with single bus system and thus impossible to be benefited from exploitation of system target architectures.

Contrary to physical prototyping, the virtual prototyping technique does not require special equipment to carry out the co-simulation for a mixed codesign system with hardware/software components and communicating links. It has the following advantages: less cost, fast turnaround time, and flexility. The core concept of virtual prototyping is *the co-simulation* of codesign system with mixed hardware/software components and communication links within a unified environment. We classify those published co-simulation methods in two categories: software-based and hardware-based. The former one is supported by the co-simulation environment written in a program language such as C or C++, which has to invoke a special simulation tool for simulations of hardware components. One such example can be found in the Ptolemy Simulation Environment [BHL+94]. The latter one, on the other hand, is based on a hardware description language. It has to call in software processes for the simulations of software components. An example of this has been reported in [TAS93]. The aforementioned method has dominated this field since early 1990s. There are commercial products available in recent years, such as the *Seamless Co-simulation Tool* produced by the *MentorGraphics* [KN97]. The major drawbacks in these special simulation tools are:

- The system target architecture is unchangeable, which is supposed the embedded system with single bus target architecture. That prevents the benefits from exploiting system target architectures.
- Software components have to be allocated to the processors specified by the tool providers, which restrains the design from adopting other processors that would be more suited.

Since commercial hardware CAD tools, particularly VHDL environments, were extensively used in late 1980s, there has been a new development in this realm. It advocates the co-simulation entirely within (V)HDL simulation environments. The difficulties now lie in the co-simulation model that can accommodate not only the communication links but also both hardware and software components. Some of the relevant publications can be found in [Nie91] [BFS94] [BE97] [FFSS97] [KMA97] [SJA93] [EPD94].

Our proposed experimental codesign methodology follows this new development in co-simulation. Further discussion in this aspect is delayed until Chapter 5 that has been assigned to grasp the relevant details.

4.2.4 Target Architecture – Single Bus vs. Multiple Buses

The target architecture must be taken into account in partitioning phase, as it has significant impact on system performance estimation and evaluation. Current system target architecture assumed in other codesign approaches is overwhelmingly the one with *single system bus*, to which are attached a general processor, hardware components, and a system memory block. This type of system target architecture has been the orthodox architecture assumed in the hardware/software partitioning research since the codesign research started. The advantages of it can be captured as follows:

- The performance estimation and partitioning algorithm are made straightforward because of the simple communication mechanism.
- The performance evaluation in physical prototyping is easy to implement and comparatively cheap.

- The general-purpose processor is naturally taken as a bus master that provides facilities for bus traffic control. The inclusion of such functionality in an ASIC chip will otherwise significantly increase the total hardware cost and the complex.
- The single level memory subsystem avoids the complexity of analyzing and synthesizing hierarchical memory subsystem.

The historical reason to involve the hardware components in an embedded system is to increase the system computation throughput. Specialized hardware components attached to a system bus can certainly speed up system execution. However, the target architecture based on this kind of structure has to suffer from the communication bottleneck inherited from the single bus system. An extensive experiment has resulted in the comprehensive conclusions [Edw97], which suggests that the ASIPs can benefit the system speedup more than ASICs do. While this suggestion is recognized as a prospective research topic, we instead offer an alternative solution in this project i.e. the system target architecture should also be exploited in support of increasing system throughput in codesign systems. One of the special target architectures has been illustrated in Figure 2.5, which as discussed earlier in Chapter 2 uses the *layered bus structure* [Sri93] to increase the communication bandwidth and system execution speed. An outstanding problem with this architecture is that it has to be simulated on the physical prototyping platform, which is, as pointed earlier, expensive and inflexible. In our codesign methodology, however, these problems are dealt with by promoting the *virtual prototyping technique*, which will be discussed later in Chapter 5.

4.3 The Partitioning Method in the Proposed Methodology

Although present research in this field varies, it can categorically be divided into the following topics: the performance estimation/evaluation, supporting tools (or environments), and partitioning algorithms. The partitioning method we proposed in this project is related to the performance estimation/evaluation and supporting tools. The partitioning algorithm remains as one of the further research topics, which will be discussed in the conclusion chapter.

Hardware/software partitioning must satisfy various system constraints that could be monetary cost, performance, circuitry size, power consumption, and so forth.

Apparently, different strategies must be adopted to meet individual system constraints. The partitioning objectives in our codesign methodology are to satisfy system timing constraints and, at the same time, to reduce as much hardware cost as possible. Both of them are paramount features in embedded real-time systems.

In the absence of partitioning algorithm, the partitioning process in our methodology has to be iterative and rely on the previous experience and the feedback from the co-simulation in VHDL environment. The partitioning approach is straightforward, which uses the profiling information obtained during the functional verification in VHDL simulation. It helps dispatch the Co-BSL primitive/class to hardware components. The criterion is that assigning those computation intensive Co-BSL instances to hardware component can undoubtedly increase the system throughput so as to satisfy the system time constraint. The selection of such instances should consider the hardware cost as well.

4.4 System Profiling with VHDL Simulations

As mentioned in section 3.6, the system profiling process adopted in our methodology is associated with the system functional verification in VHDL simulations. It requires additional VHDL statements inserted into the VHDL program that is converted from its Co-BSL specification. These extra VHDL statements are aimed at collecting the following simulation statistics when their host VHDL program is executed for functional verification:

1. Computation load of basic modules (*loop, subprogram, or process*), according to their activation occurrences
2. Communication intensity of communication paths, in total number of (a)synchronous sending or receiving operations

The first statistic can obviously help identify the computationally intensified modules (the process particularly concerned here) that are appropriate candidates to be dispatched to hardware implementations. The second one assists in allocation of hardware/software components to a particular system bus that is included in the assumed system target architecture.

In general, the partitioning process in our methodology comprises the following tasks:

1. VHDL simulations for functional verification and system profiling:
   (i) converting the Co-BSL specification into VHDL program
   (ii) adding the extra VHDL statements to record the profiling information
   (iii) simulating the VHDL program obtained in (ii) in a VHDL simulation environment
2. selecting a system target architecture and the connections between hardware/ software components
3. dispatching time critical components to hardware implementations
4. analyzing the feedback from the performance evaluation and repeating tasks 2 and 3

In the iterative partitioning process outlined above, the conversion from Co-BSL to VHDL program has been discussed in Chapter 3 and the work 2, 3 and 4 shall be the focus of Chapter 5, which copes with the system performance evaluation i.e. *design feedback* in Figure 4.2. Presented in this section, is the collection of profiling information of a codesign system in VHDL simulations. The time critical system components will accordingly be identified and dispatched to hardware implementations. The example 4.1 below serves as a demonstrative example.

Example 4.1- Arithmetic Coding System

This example demonstrates the acquisition of profiling information from VHDL simulations. The codesign system undertaken is *Arithmetic Coding System* [BCW91].

The arithmetic coding is widely used for information compression, which encodes messages in an interval of real numbers between 0 and 1. While the message becomes longer, the interval required to represent it becomes smaller and the number of bits required to specify that interval grows. In other words, it replaces a stream of input with a single floating-point number as output. This number can be uniquely decoded to create the precisely same stream of message that comes through its original construction. Although the concept has been known for a long time, only in recent years were practical methods established to implement arithmetic coding on computers with standard integer math. This is because floating-point math is not feasible in practice. Further technical details can be found in [NG97].

The following piece of pseudo-code summarizes its encoding and decoding algorithms.

```
/*  ARITHMETIC ENCODING ALGORITHM                                   */
/*  Ensure that a distinguished "terminator" symbol is encoded last, then   */
/*  send out any value in the range [low, high].                    */

EncodeSymbol (symbol, CumProb)
    range := high – low;
    high  := low + range*CumProb[symbol – 1];
    low   := low + range*CumProb[symbol];

/*  ARITHMETIC DECODING ALGORITHM                                   */
/*  Value is the number that has been received                      */
/*  Continue calling DecodeSymbol until the terminator symbol is returned.  */

DecodeSymbol (CumProb)
      find  symbol such that
          CumProb[symbol] <= (value – low) / (high – low) < CumProb[symbol – 1];
          /*  This ensures that value lies within the new [low, high) range   */
          /*  that will be calculated by the following lines of code.         */
      range := high – low;
      high := low + range*CumProb[symbol – 1];
      low  := low + range*CumProb[symbol];
      return symbol;
```

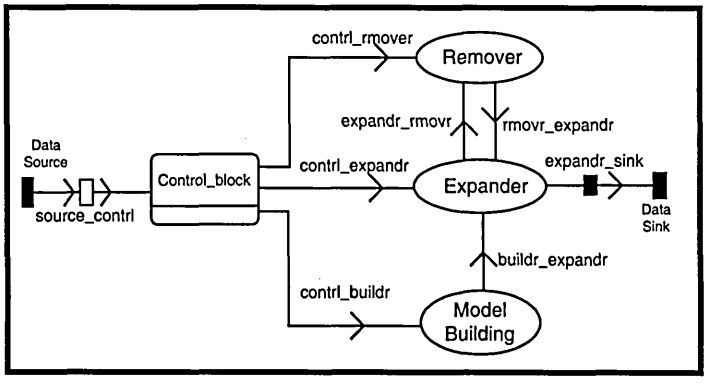


Figure 4.3 Arithmetic Decoding

The codesign system depicted in Figure 4.3 is an arithmetic decoding system. Its arithmetic encoded information has been created in advance and stored in a file, which then feeds the system through the *Data Source* as stimuli. Notice that the stimuli are encoded messages and the originals are unavailable at the decoding site. The decoding process, however, needs the statistical information created during encoding process, such as CumProb[symbol] and CumProb[symbol – 1] as used in the *Decoding Algorithm*. Although an alternative coding scheme called *Adaptive Arithmetic Coding* provides a solution to this problem, it complicates the matters unnecessarily in this example. We therefore assume that the statistical information is added before the encoded information. The process *Model Building* in Figure 4.3 is designed to receive

the statistical information and build its model. The process *Control_block* is needed to identify different types of information and dispatch them to relevant processes. Whereas the process *Expander* retrieves the statistical model established by the *Model Building* and expands the original information gradually, the process *Remover* assists this procedure by decoding compressed bits from the *Control_block* and sending back decoded message back to the *Expander*. The expanded message is preserved in the external process *Data Sink*.

Extra VHDL statements have been added to the VHDL program for collecting the system profiling information.

For example, the following VHDL statement lines are used to lodge the communication load from the process *Control_block* to *Expander*.

```
--/ recording the communication from Control_block to Expander /--
  if contrl_expandr.status = active_source then
    tmp := contrl_expandr.color.data2;
    write(l, tmp, right, 1);
    temp := temp + 1;
    if temp > 15 then
      write(l, NOW, right, 15);
      writeline(datafile,l);
      temp := 0;
    end if;
  end if;
```

Besides, the following VHDL statements register the invocation time for the process *Control_block*.

```
--/ counting invoking time /--
  write(l, t_times, right, 1);
  t_times:= not t_times;
  if t_count >= 29 then
    write(l, NOW, right, 15);
    writeline(datafile,l);
    t_count:= -1;
  end if;
  t_count:= t_count + 1;
```

System profiling information acquired from VHDL simulations has been attached to the process graph in Figure 4.3 and shown in a new process graph illustrated in Figure 4.4. The profiling information is divided in two groups: the invocation time for each process

and the communication overhead for each communication channel. One can easily recognize that the *Expander* is most active process invoked for 570 times in total and the *expandr_rmover* is the busiest channel, through which 65.5-KB's message has passed.

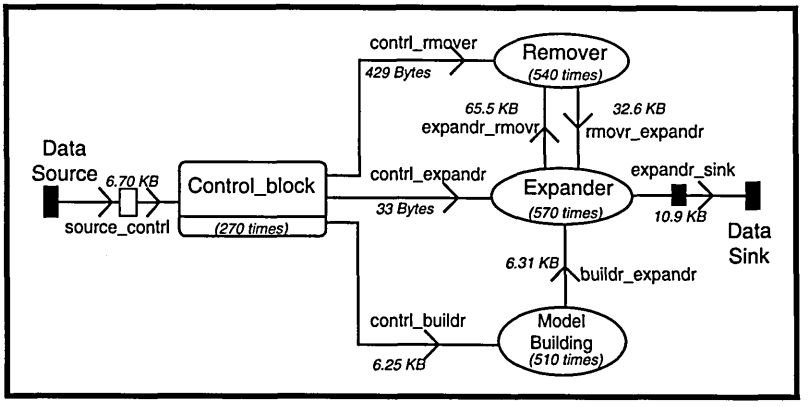


Figure 4.4 Process Graph Showing Simulation Results

Individual VHDL program plus simulation results has been listed in Appendix D for reference.

4.5 Partitioning and Component Allocation with Multiple Buses

Despite similarities between hardware/software partitioning and the partitioning of either pure hardware or software system, the former represents a rather complex task due to its heterogeneous nature. Moreover, the introduction of distributed target architecture apparently complicates the matter further. For example, in *Arithmetic Coding System* described in section 4.4, if a target architecture with two system buses is chosen, six system's processes (including two external entities) could have 2^6 assignment schemes. As the number of component increases, the number of alternatives will explode, not to mention another explosive dimension in which the system components can be allocated to either hardware or software implementation.

The philosophy in previous codesign methodologies is simple, i.e. to dispatch computationally intensive component to hardware implementation in order to achieve high system performance. There was no question of exploitation of component allocation since only one system target architecture with single bus was available and all components had to be linked to this system bus. However, hardware implementation is only one of many solutions to improving system performance. One of the major contributions from this thesis is to have addressed this shortcoming and provided a

platform (detailed in Chapter 5 and 6) for the codesigner to experiment with both hardware/software partitioning and the allocation of component in the interest of system performance.

Based on this new platform, the exploration of the design space extends beyond the partitioning of hardware/software components. A new dimension is now established, in which the system components are not only assigned to hardware/software implementations but also dispatched to the appropriate places within the system target architecture, i.e. in addition to being dispatched to hardware/software implementation, a component can also be allocated to a specific bus layer in order to achieve high system performance.

As stated earlier, this project is not aimed at automatic partitioning process. A heuristic approach is instead adopted to facilitate the partitioning and allocation. With regard to hardware/software partitioning, this research has been focused on:

- Analysis of previously established partitioning methods
- Identification of the shortcomings and problems in this domain
- Proposal of solutions to those problems

While these tasks are mostly accomplished, we have left an advanced topic for future research. It is to improve the system performance by exploiting the combination of hardware/software partitioning and allocation of communication channels, which will be one of the topics discussed in Chapter 7.

Chapter 5

The Performance Evaluation

This chapter deals with the system performance evaluation for codesign system. In our project the system target architecture with layered bus communication structure is used as a platform, on which the performance evaluation for codesign system is carried out. In addition to the architecture model, we have also provided the implementation method and the supporting tools in VHDL packages and libraries.

The content of this chapter has been published in [CRL00] in an abridged version. This chapter will detail the architecture model and the performance evaluation method developed in our project. The chapter begins with a brief review on both emulation and simulation techniques that are both widely utilized in the codesign school. The co-simulation technique and virtual prototyping are highlighted. Following this is the rationale for the co-simulation and virtual prototyping technique, proposed in this research project. In addition, the conversion from high-level system co-specification to its low-level implementation is introduced and the bus protocol and VHDL packages plus libraries are described in the following sections. Finally the co-simulation to evaluate the system performance for codesign system is presented.

5.1 Performance Evaluation Techniques for Codesign

Improving system performance is an essential issue in the codesign of hardware/software. It is a critical goal in codesign of hard real-time embedded system. The system performance can be evaluated in the following three ways: implementation, emulation, and simulation. The implementation is apparently less attractive due to its huge cost and inflexibility. It is therefore not in the interest of this research. While the emulation is carried out on its physical prototype platform, the simulation is instead executed in software simulators. In the literature of codesign, this kind of simulation is often referred to as *co-simulation* since it deals with heterogeneous systems with both hardware and software components inside. Unlike a pure software or hardware system, the system performance now involves both individual hardware and software

components and the communications that pass through homogeneous/heterogeneous components.

In the codesign discipline, a great deal of effort has traditionally been spent on the emulation technique of performance evaluation. One of the good reasons for this is that a codesign system can be emulated on its prototype target architecture, which is generally composed of microprocessors, ASIC (*Application Specific Integrated Circuits*) components and buses. Hardware and software components communicate via a global memory as a rendezvous. This approach has become feasible thanks to the programmable hardware components, such as FPGA (*Field Programmable Gate Arrays*), which are now commercially available [FPGA00]. In FPGA, like its software counterpart, the hardware components can be programmed and the trade off in hardware and software becomes feasible. As a result, the system performance can be evaluated by the emulation on its physical prototyping platform. The emulation could predict accurate system performance but it is expensive and inflexible especially in terms of the system target architecture. It can in fact proceed only when the prototyping hardware components are available. In addition, the programmable hardware has various constraints, such as size and interface, and it is therefore not always readily fitted in the real system [IEEE92].

In contrast to emulation, the software simulation has the following variety of advantages: low cost, flexibility, and short design turnaround time. Because of these advantages, there has been a growing interest in co-simulation techniques. A part of relevant publications in this realm can be found in [WDW94][BFS94][BFS96][BE97] [PLCV97][HB97][LLV98][LLS99]. The co-simulation can be either software-based (in C/C++ program) or hardware-based (in VHDL or other HDL simulators) but neither of these two approaches can alone undertake the task. The software-based co-simulation favours functional verification of individual software components but behaves awkwardly when dealing with the performance of hardware components and the communications between heterogeneous components. On the other hand, the hardware-based co-simulation favours the performance evaluation for individual hardware components but no mechanism supports the evaluation of those software counterparts and the communications. Another drawback of the hardware-based co-simulation lies in

its simulation efficiency (simulation performance itself). As the density of hardware circuit increases, driven by advancing technologies, traditional event-driven hardware simulators become increasingly incapable of responding to a huge number of events, which would have to take hours or even days to complete the simulation. To achieve high-performance solutions, EDA vendors started to switch the interest to a new technique, *the cycle simulation*, which does not take account of the detailed circuit timing but computes the steady state response of the circuit at each cycle boundary [Bha98]. On the other hand, the hardware-based co-simulation at the behaviour level (i.e. high level) is not prone to this problem because of dramatically decreasing number of event at this level.

Notwithstanding some of co-simulation tools which emerged during late 90's, such as the *Seamless* from the MentorGraphics [KN97], the system target architectures they supported are primitive, all confined in the target architecture with single bus communication system. It could easily fall into the communication bottleneck problem inherited by this type of target architecture.

5.2 Justification for the Proposed Performance Evaluation Method

Within our methodology, we proposed a co-simulation technique [CRL00] that encompasses the following:

1. The virtual prototyping model with *layered bus architecture*, which enables system target architecture to be exploited as a new dimension to improve system performance
2. The synthesis method, which maps the high-level co-specification to the low-level implementation based on the target architecture indicated above
3. The asynchronous generic bus protocol plus its standard bus interface modules and the VHDL packages & libraries, which have further materialized the proposal

The contents relevant to the bus protocol and VHDL packages and libraries are being discussed in the following sections because of their complexities.

The significance of these features can be justified as follows. First, the system target architecture of an embedded system has undergone consecutive changes as a result of technology advance in hardware/firmware manufacture. Modern embedded systems are

no longer bounded by the single bus target architecture. A typical example can be found in [SB91] and also referred to Figure 2.5. It has the target architecture with 4 layers. The bottom two layers are custom boards, each having one or more programmable processors. Each processor in turn coordinates a number of application specific slave modules which can be either hardware or software components. This type of target architecture employs the hierarchical bus organization that increases the communication bandwidth. While it provides flexibility and scalability, the system performance evaluation has to rely on the emulation technique, which is costly and some times impossible before all the hardware components are constructed. Our project avoids this problem by employing the virtual prototyping plus co-simulation technique.

Secondly, in codesign community, the current ad hoc transition from high-level co-specification to low-level implementation presents a major failure from *Software Engineering*'s point of view. The well-defined system structure created at the co-specification phase has broken down in the low-level implementation and no trace of object-orientation remaining at this low level, which will induce troubles when the codesign system is maintained at a later stage. Our research, however, tackles these problems by introducing the consistent system synthesis method, which maps a high-level co-specification to its low-level implementation with the property that the object-based properties in the system co-specification phase are conceptually maintained.

As discussed in Chapter 1, the system target architecture previously adopted by other researchers is, by default, the one, in which a single synchronous system bus undertakes communications between hardware and software components. In this research, however, we pursue the communications in different ways. First, an asynchronous bus protocol is designed instead of a synchronous one, since hardware/software components in codesign system are reasonably expected to be clocked independently (*asynchronous*) and the number of components may change in line with the requirement of cost-effective system performance. The asynchronous communication bus is well suited to this environment. Second, the layered multi-bus architecture is to be trailed to test the feasibility both in terms of conceptual model and the viability of co-simulations in VHDL environment.

It should be pointed out here that the performance of a codesign system is mainly concerned with its communication cost and the performances of individual components. The structure of system target architecture certainly influences the analysis of communication cost that is counted in the whole system performance together with the performances of individual hardware/software components. At present some researchers study the communication cost of codesign system by using statistical analysis based on mathematical models [HB97][BKK+99] [KM98]. They provide the potential of automatically estimating the communication cost and supply the heuristics in the hardware/software partitioning phase. This research, however, is focused upon the experimental perspective rather than statistical models. We prefer this approach because of the following observations:

- Most codesign systems are hard real-time embedded system that is governed by strict timing requirements. Co-simulation techniques could provide reliable and accurate prediction in respect of system timing.
- The statistical analysis is proven to be effective in designing a general distributed network system such as the Internet that incorporates communications via relay mechanism through communication routes [CDK01]. However, it would make the system performance difficult to evaluate if the co-simulation model was based on the changeable or uncertain transmitting routes. On the other end of the spectrum, target architectures in codesign system are static and so are the communication routes, which indicates the communication in codesign system is relatively "well-behaved" and thus able to be simulated based on its target architecture model.

One of the major contributions from our methodology is that in addition to the trade-off in hardware/software implementations, a new dimension which is the trade-off in system target architecture has been employed to improve the system performance in codesign. The orthodox single-bus system target architecture commonly used in codesign discipline has its fundamental flaw that is the communication bottle-neck problem. Although the target architectures proposed in our methodology are relatively simple, for the first time in codesign discipline, however, it allows the user to exploit both hardware/software implementation and the system target architecture, which increases the system throughput and hence improves the system performance. A

comprehensive example of improving system performance by the exploration of system target architecture will be demonstrated in Chapter 6.

5.3 Layered Bus Prototyping Model

Figure 5.1 illustrates our prototyping model, in which the layered system buses with diversified throughput undertake the communications in the system. The reason this type of system model is selected is that it is straightforward and easy to implement in a VHDL program. In addition, the communication throughput of this model is readily extendable in adding more bus layers and/or designating different bus performances. Based on this system model, hardware/software components are connected to different buses with certain criteria so as to satisfy the system performance and/or hardware costs.

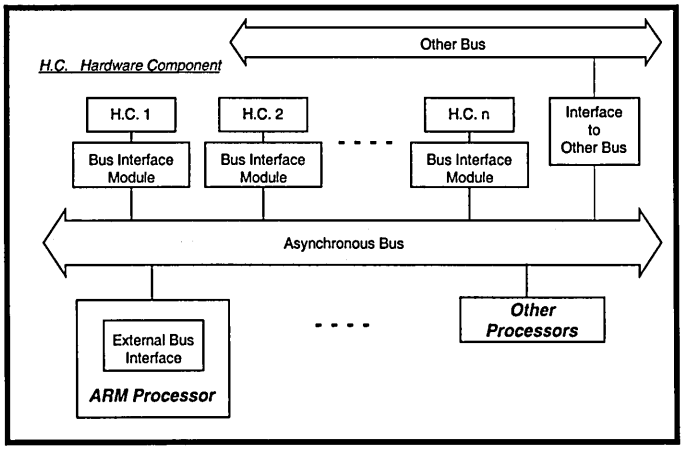


Figure 5.1 Distributed Target Architecture with Layered Bus Structure

An extra bus interface component is designed to provide communication gateway over buses [Kai93][Dav+94]. Each hardware/software component can only connect to a system bus through *Bus Interface Module* (BIM) that has been designed and stored in a VHDL library.

Although other communication mechanisms, such as direct hard-wired connections or interrupts, are also viable in this model, they are not included at present for the simplicity reason. In the diagram, we assume that the components destined for software are allocated to ARM processors [Fur96][Som93] but it is not the prerequisite in this model. Other embedded processors are also fit for the purpose. The reasons for adopting the ARM processor arise from its software development environment i.e. the ARM SDT

[ARM97] available and its reputation for low power consumption in the embedded system designs. In this module, the interface connection of software/hardware components is unified. It is depicted in Figure 5.2 and 5.3. The internal details in the module will be introduced in the next section.

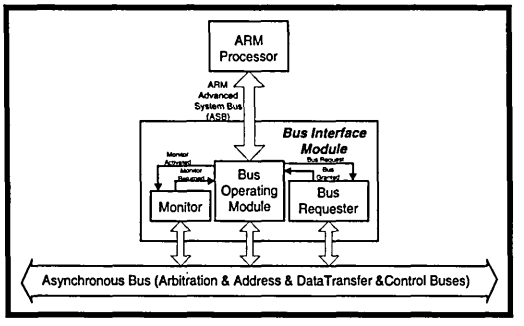


Figure 5.2 Interface for the ARM

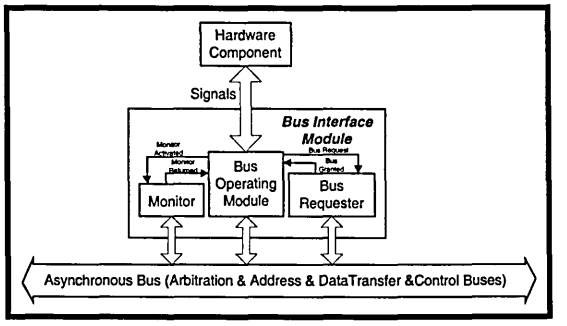


Figure 5.3 Interface for Hardware Component

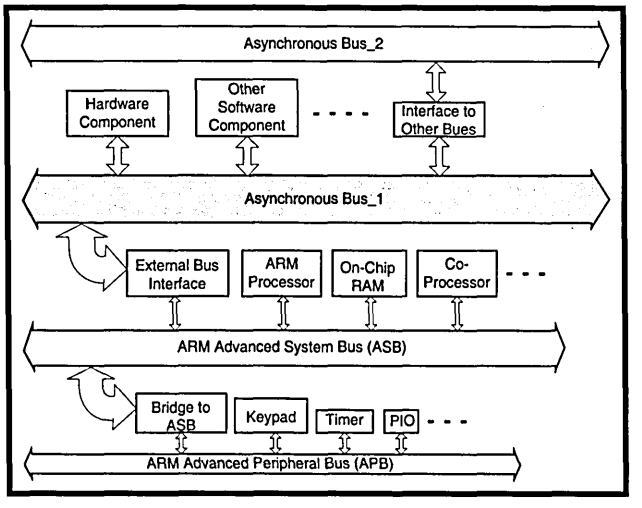


Figure 5.4 Layered Bus Structure

Figure 5.1 can be further refined in Figure 5.4, added internal details of ARM processor. The ARM processor contains two on-chip buses. They are *Advanced System Bus* (ASB) and *Advanced Peripheral Bus* (APB). The ASB is the main high-speed backbone communication bus and the APB the low-speed and low-power peripheral bus. Besides, there is a component, namely *External Bus Interface* (EBI) connected to the ASB, communicating reciprocally with other components outside the ARM processor. In our model, it has been replaced in the *BIM*, which is now a standard bus interface module. One can see from the Figure 5.4 that the layered asynchronous buses plus the on-chip buses inside the ARM processor do constitute a flexible communication model of codesign system's target architecture.

5.4 Asynchronous Bus Protocol and Bus Interface Module

Since this section is not intended to serve as a general introduction to bus protocols, it is appropriate to only enclose most important and relevant details in the following content. Those excluded can be referred to [Gia90], [VME82], and [Tex88]. Although it adopted the common handshaking protocols previously published in other bus protocols, the asynchronous bus protocol presented in this section is a new bus protocol specially designed for the purpose of exploration of system target architecture to improve the system performance. The differences from other bus protocols will be explained where approperait.

The asynchronous system bus consists internally of four parallel sub-buses. They are *Data Transfer Bus* (DTB), *DTB Arbitration Bus*, *Control Bus* (CB), and *Address Bus* (AB) respectively. Figure 5.5 depicts the internal details of the *bus interface module* that is an interface for connections between the sub-buses and hardware/software components. The module is composed of three sub-components: *Bus Requester*, *Bus Operation Logic*, and *Bus Monitor*. They communicate with each other through internal signals. While the bus requester is connected to the arbitration bus to apply for control of the DTB, the bus operation logic and the bus monitor are all connected to the DTB, CB, and AB to complete handshaking and data transfer. This module along with other modules introduced in the next section has been developed and integrated into a VHDL library as the components to be integrated in user's VHDL simulation programs. The performance evaluators are therefore relieved from dealing with the complex handshaking protocols.

The arbitration bus uses the following lines to request and grant DTB bus:

- BR16 (bus request lines, 16 bits)
- BG16 (bus grant lines, 16 bits)
- BBSY (bus busy line, 1 bit)

The width of arbitration bus indicates how many legitimate bus contenders can exist Sixteen bus masters, in this case, are allowed and each request line has a corresponding grant line.

The DTB, AB, and CB are defined as follows:

D32 (data path width: 32 bits)

A32 (address bus: 32 bits)

- A16 (address path width: 16 bits)
- S16 (segment path width: 16 bits)

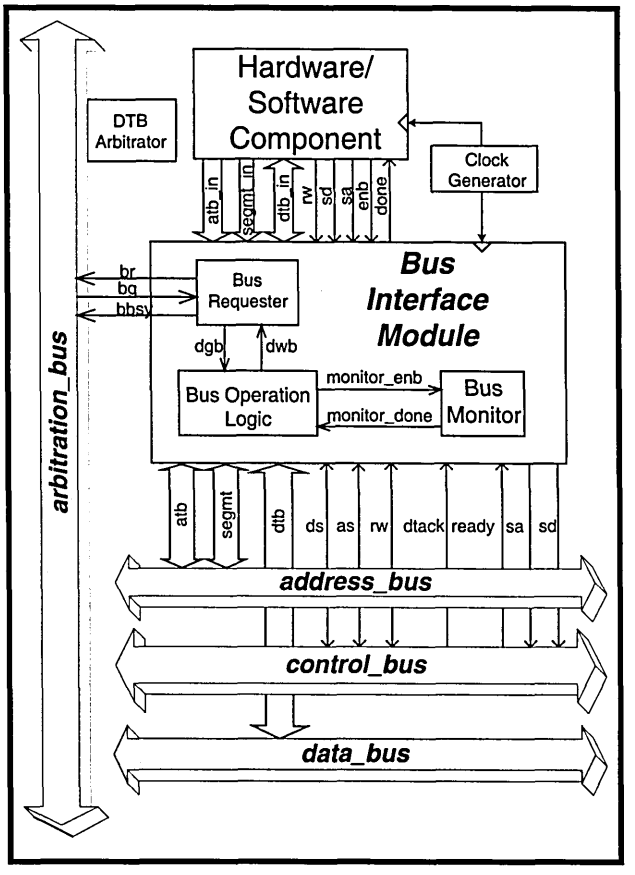


Figure 5.5 Schematic of the Bus Interface Module

C7 (Control Bus: 7 bits)

- c1: AS (address strobe)
- c2: DS (data strobe)
- c3: W/R* (read/write select)
- c4: DTACK (data acknowledge to master)
- c5: READY (indication of the transfer complete)
- c6: S/D* (indication of the DTB transfer through same or different bus)
- c7: S/A* (indication of the DTB transfer blocked or non-blocked)

Notice that apart from normal control lines typically adopted in other bus protocols two extra control lines are defined here to clarify the following positions:

1. whether a DTB transfer occurs on the same/different bus ... (line *sd* in Figure 5.5)

2. whether a DTB transfer is blocked/non-blocked ... (line *sa* in Figure 5.5)

The address bus includes both address path and segment path. The latter is used for future extension of bus layer and system functionality. Its usage in this prototype will be reflected as the synthesis of the prototyping model is discussed in the next section.

5.4.1 DTB Acquisition

A potential DTB master has first to request the arbiter to grant its use of DTB and then controls the DTB for data transfer. A *Round Robin Select* arbiter is chosen so that each master connected to the DTB has same priority. This policy could be accordingly altered to satisfy different priorities in different components to meet system constraint, but we have left this topic as a future research topic.

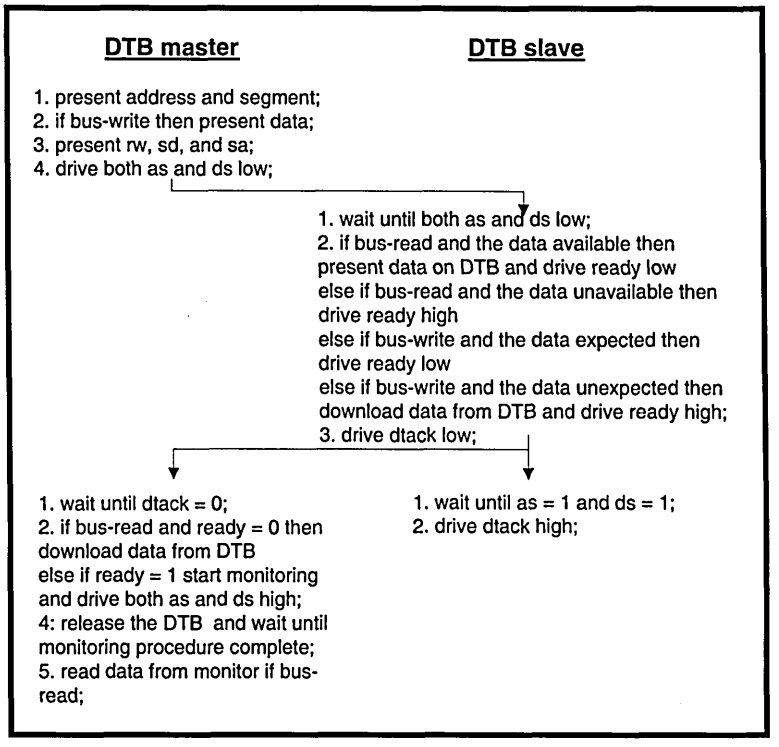


Figure 5.6 DTB Handshaking Procedure

The low-level handshaking protocol for DTB arbitration is similar to those of VME's. Figure 5.6 serves as an example of the handshaking procedure for bus arbitration and operation. It demonstrates two potential bus masters A and B competing for control of DTB. Internal signals, dwb and dgb represent the statuses *device wants bus* and *device is granted bus*. Three external signal lines br, bg, and bbsy are used for bus arbitration handshaking. Assume, in this example, that master B is first granted the DTB and completes data transfer. The Bus control is then handed over to master A that finishes data transfer after obtaining DTB control.

5.4.2 DTB Operation

A typical DTB handshaking procedure is shown in Figure 5.6. Whereas its further details have to be explained when combined with the synthesis of the prototyping model, a general introduction of the procedure is given in Figure 5.7.

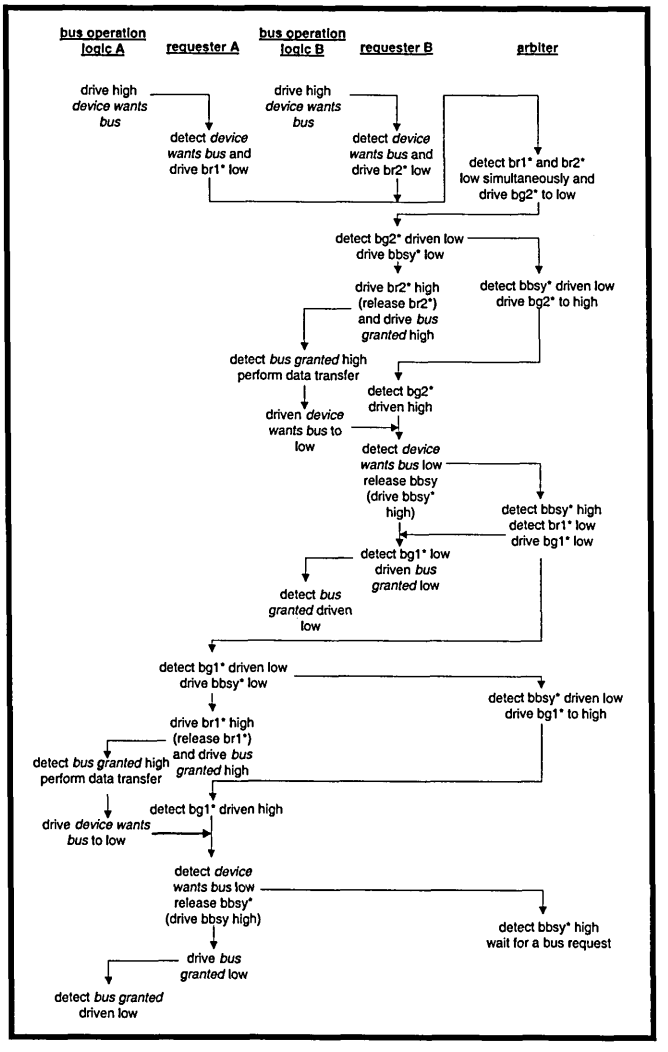


Figure 5.7 Typical DTB Arbitration Procedure

The DTB handshaking is initiated by the bus master and replied by the bus slave. After obtaining control of DTB, the master presents address signal on AB and data on DTB if it is a bus writing operation. It then waits. After receiving the transfers acknowledge from the slave, the master terminates the data transfer session by raising the *as* and *ds* up to '1's. The asynchronous nature of the DTB allows the slave to control the amount of time taken for the data transfer. This feature has to be exploited with caution to prevent component from holding up the DTB due to other party's unreadiness or to avoid deadlock. If, for example, several synchronous communication channels are allocated on the same system bus, the synchronous communication could result in holding up the

system resource (in this case the system bus) for a long time or even forever (deadlock) [And95]. In our protocol, therefore, the master is required to release the DTB no matter whether the data transfer can complete. This feature is supported by a global memory and a monitor, which are mentioned latter and fully discussed in the next section while the synthesis of prototyping model is tackled.

5.4.3 DTB Monitor

Each bus interface module maintains regular surveillance on the system bus. While bus monitoring is auxiliary in other bus protocols, it is a necessity in our protocol, which prevents the DTB being held for a long time or even forever (*deadlock*) while components communicate through those communication channels defined in Co-PARSE specification and wait the other side of the communication channel to response. More details will be given latter in the relevant places.

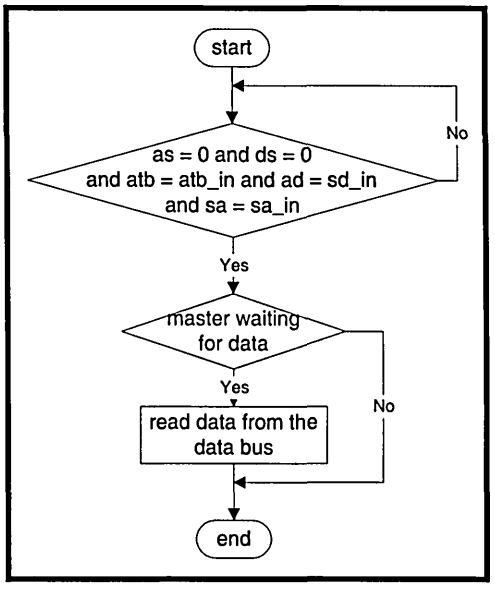


Figure 5.8 DTB Monitoring Procedure

The monitoring procedure is illustrated in Figure 5.8. If a bus master can not get satisfactory result from the bus slave, the bus is immediately released and the monitoring procedure is triggered. Three unsatisfactory results could start this procedure:

a) synchronous channel's sender blocked at the rendezvous because of the receiver unready
b) synchronous channel's receiver blocked at the rendezvous because of the data unavailable

c) asynchronous channel's receiver blocked at the rendezvous because of the queue empty

Notice that only case b) and c) actually receive data from the DTB. Before entering this procedure, the DTB has been occupied by other master through arbitration. We will elaborate on this when the synthesis of prototyping model is introduced.

5.5 Synthesis of the Prototyping Model

As mentioned in previous chapters, during the hardware/software partitioning phase, processes specified in the extended PARSE Process Graph and the Co-BSL program are assigned to either hardware or software components. Software components are allocated to ARM processors while hardware components are further refined in VHDL description and automatically synthesized into VLSI chips by synthesis tools. The remaining task now is to furnish communication channels with prototyping components, i.e. mapping the communication channels to the physical prototyping model, which, in this case, is the layered bus architecture. To demonstrate feasibility, we focus on mapping communication channels to the target architecture with up to two asynchronous system buses, although more bus layers are theoretically addible. The following discussion is therefore divided in four topics that are denoted by the combination of control lines sa and sd.

- synchronous channels on the came bus (sa = 0 and sd = 0)
- asynchronous channel on the same bus (sa = 1 and sd = 0)
- synchronous channels with two buses (sa = 0 and sd = 1)
- asynchronous channel with two buses (sa = 1 and sd = 1)

5.5.1 Synchronous Channels on the Same Bus

The key to implement synchronous channels lies in the employment of a global memory [LW97] as the rendezvous for both the sender and the receiver of a synchronous channel. Without loosing generality, we can assume that the message passing through the synchronous channel is in integers each with 32-bit length. Other data types can always be represented by integers. In addition, each synchronous channel is assigned a number. Both sender and receiver place this number to the address bus when they communicate. Obviously, all synchronous channels allocated on the same system bus can be organized in a chunk of memory as shown in Figure 5.9. The global memory has been enlarged in Figure 5.10 and each channel is allocated a 32-bit memory location as

data storage and 1-bit mark as an indication of either the sender waiting for message to be accepted or the receiver waiting for the message sent to the channel.

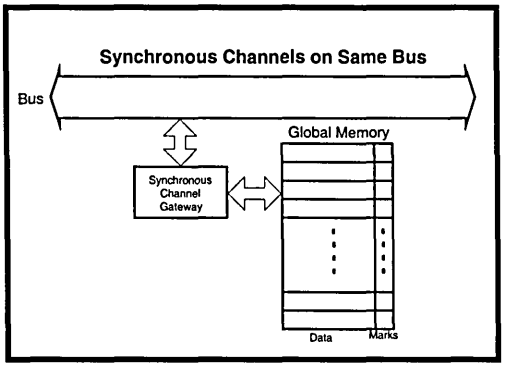


Figure 5.9 Synchronous Channels

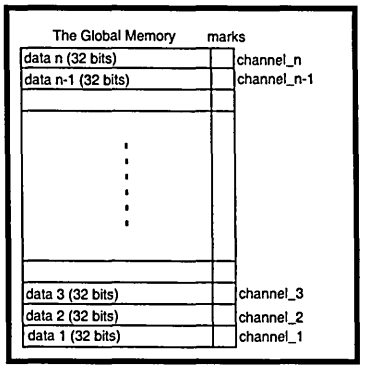


Figure 5.10 Synchronous Channels' Memory

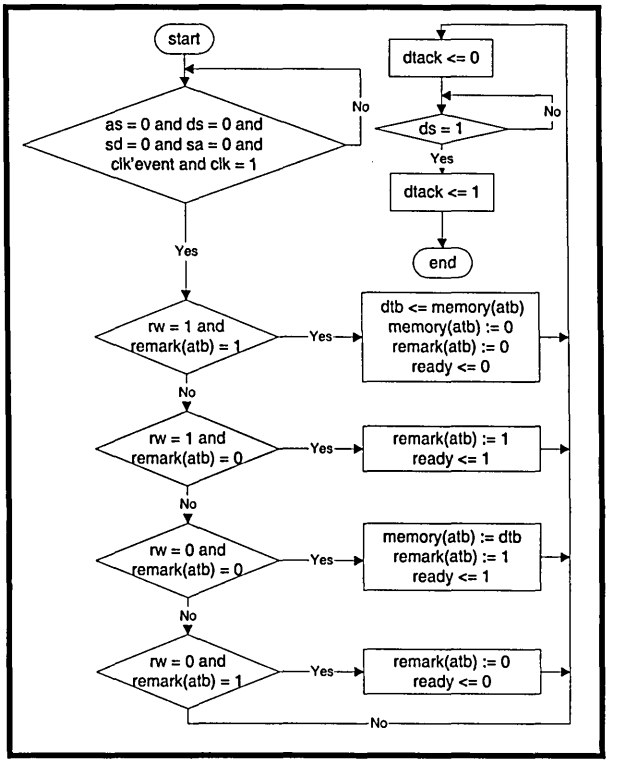


Figure 5.11 Synchronous Channel Gateway

The component *synchronous channel gateway* described in Figure 5.11 is attached to system bus and appointed as a bus slave. Its responsibilities include DTB data transfer handshaking, data storage, and bookkeeping. The first two are easy to understand but the last task needs explanation. A synchronous channel can only encounter one of the situations below:

- sender side:
  1. sending message while receiver is waiting (rw = 0 and remark(atb) = 1)
  2. sending message while receiver is unready (rw = 0 and remark(atb) = 0)
- receiver side:
  3. reading message while it is available (rw = 1 and remark(atb) = 1)
  4. reading message while it is unavailable (rw = 1 and remark(atb) = 0)

These four situations are handled correspondingly as follows:

1. remove the mark and set ready to 0
2. establish the mark, retain the message in the memory, and set ready to 1
3. place the message on the DTB, remove the mark, and set ready to 0
4. establish the mark and set ready to 1

With regard to the foregoing introduction to the DTB data transfer protocol, a sender or a receiver shall start its monitor if the receiver is unready or the message is unavailable. Hence, there is no need for a sender to write the message into memory while the receiver is waiting for it at the address because the receiver's monitor will pick up the data from the DTB automatically through surveillance. On the other hand, a receiver does not need to inform the sender after the message has been taken out of the memory because the sender's monitor can observe the movement of the message through its own monitoring procedure. This treatment immensely reduces the DTB traffic and improves the efficiency of system communications.

5.5.2 Asynchronous Channel on the Same Bus

In asynchronous communication, the sender is never blocked because it can preserve the massages in a queue, whereas the receiver has to wait if the queue is empty. At the centre of this type of communication is therefore the queue that accommodates messages in series from the sender. The implementation has been delineated in Figure 5.12 and the global memory is also expended in Figure 5.13. The queue is constructed in a block of global memory and organized as a circular storage as shown in Figure 5.13. Unlike the synchronous channel, only one mark (bit) is now required to indicate if the receiver has visited and been waiting at the queue due to the message unavailable.

The function of the component *asynchronous channel gateway* is displayed in Figure 5.14. Besides accomplishing DTB data transfer handshaking, it undertakes four tasks:

1. Establish mark and set ready to 1 if the bus master receiver finds the queue empty.
2. Take the head of the queue out off queue and set ready to 0 if the data is available.
3. Set both mark and ready to 0 if the sender finds mark has been set up.
4. Massage is entered into the queue and set ready to 0 if queue is not empty of the data not yet wanted.

They correspond to four selections and their processes shown in Figure 5.14.

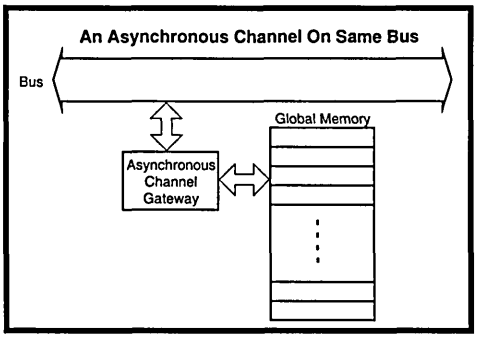


Figure 5.12 An synchronous Channel

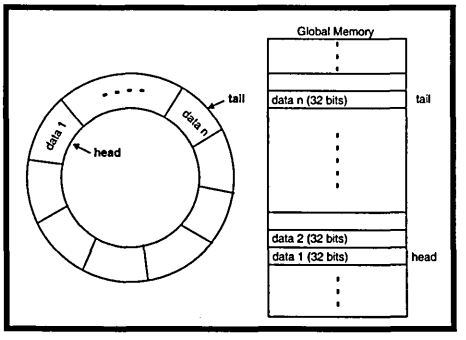


Figure 5.13 Asynchronous Channel's Memory

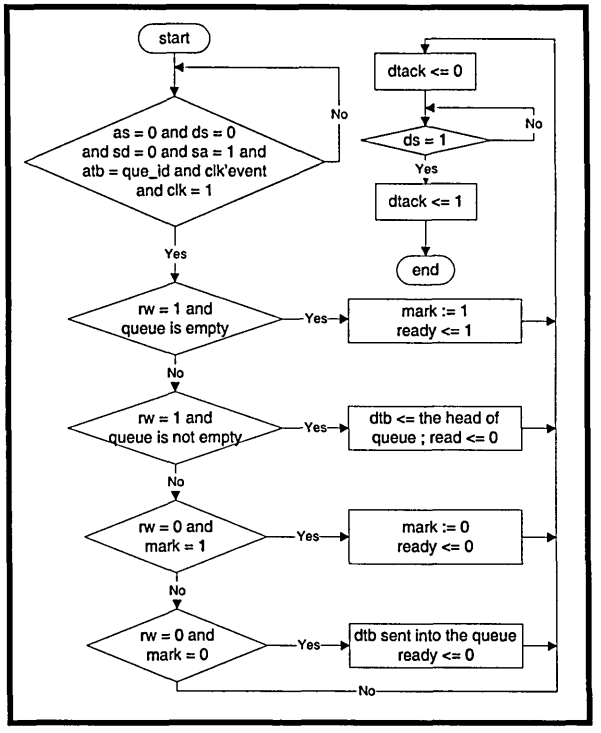


Figure 5.14 Asynchronous Channel Gateway
(*on the same bus*)

5.5.3 Asynchronous Channel with Two Buses

With two system buses, we assume that the sender of an asynchronous channel is allocated to one bus and the receiver to the other. They have to negotiate with one bus for write operation and another for read operation. Figure 5.15 exemplifies the sender of an asynchronous channel being situated on bus_1 and the receiver on bus_2. Its opposite allocation is not discussed because same component can be used, except connections to its symmetric bus signals.

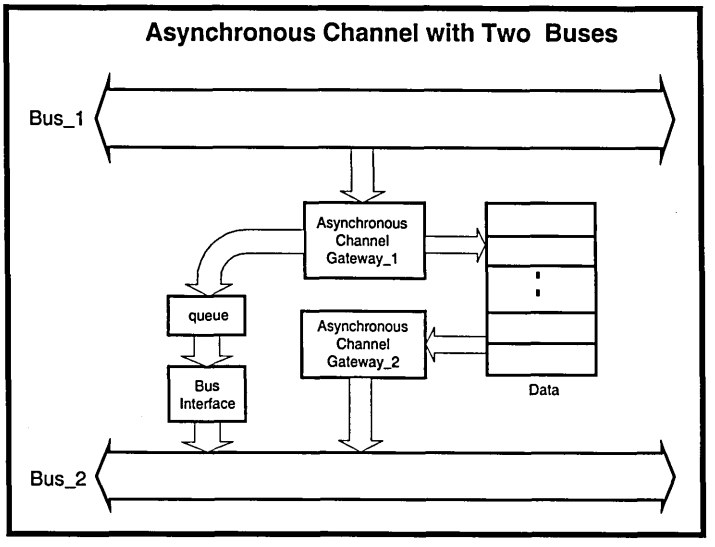


Figure 5.15 Asynchronous Channel with Two Buses

Two queues are in fact needed for this implementation. The main queue is connected to both buses through *asynchronous channel gateway_1* and *asynchronous channel gateway_2*. Its memory organization is similar to the counterpart in the asynchronous channel on same bus but it is noteworthy that the gateway_1 is write-only and the gateway_2 read-only with regard to the main queue. Besides, two channel gateways are now involved in dealing with respective DTB data transfer handshaking and main queue's bookkeeping. Another queue is instead connected to bus_2 via the *bus interface*. It feeds the bus interface with data from the gateway_1. Figures 5.16 and 5.17 exhibit the internal structures of both gateways concerned. Figure 5.18 shows the bus interface.

Two queues are in fact needed for this implementation. The main queue is connected to both buses through *asynchronous channel gateway_1* and *asynchronous channel gateway_2*. Its memory organization is similar to the counterpart in the asynchronous channel on same bus but it is noteworthy that the gateway_1 is write-only and the

gateway_2 read-only with regard to the main queue. Besides, two channel gateways are now involved in dealing with respective DTB data transfer handshaking and main queue's bookkeeping. Another queue is instead connected to bus_2 via the *bus interface*. It feeds the bus interface with data from the gateway_1. Figures 5.16 and 5.17 exhibit the internal structures of both gateways concerned. Figure 5.18 shows the bus interface.

Compared with the asynchronous channel on same bus, one of the four situations may occur during the asynchronous communication across different buses:

1. The sender is sending message while the receiver is not waiting for it.
2. The receiver is demanding message while the queue is not empty.
3. The sender is sending message while the receiver is waiting for it.
4. The receiver is demanding message while the queue is empty.

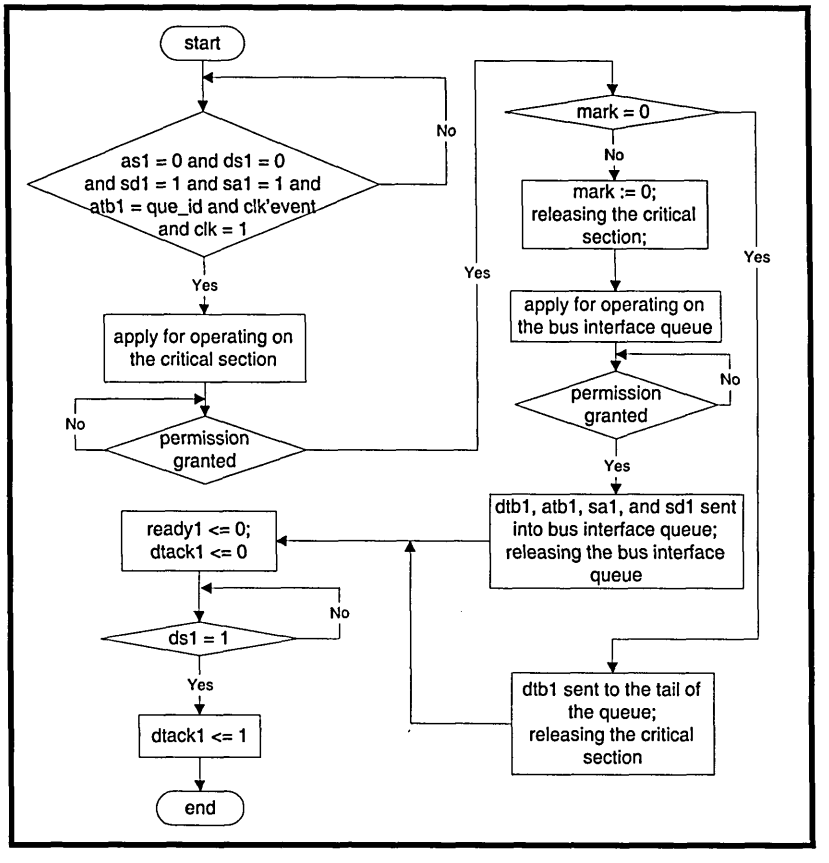


Figure 5.16 Asynchronous Channel Gateway_1 (*for two buses*)

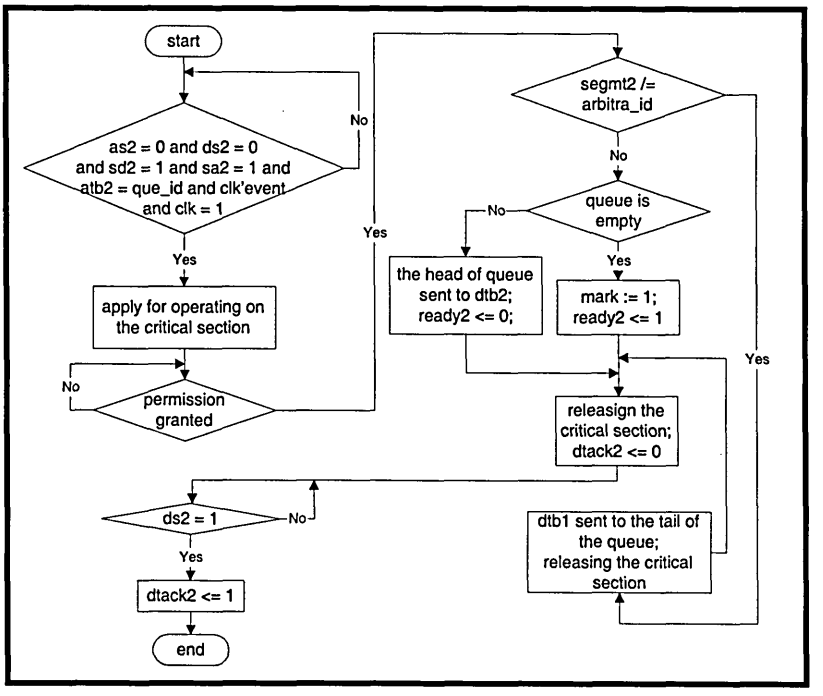


Figure 5.17 Asynchronous Channel Gateway_2 (*for two buses*)

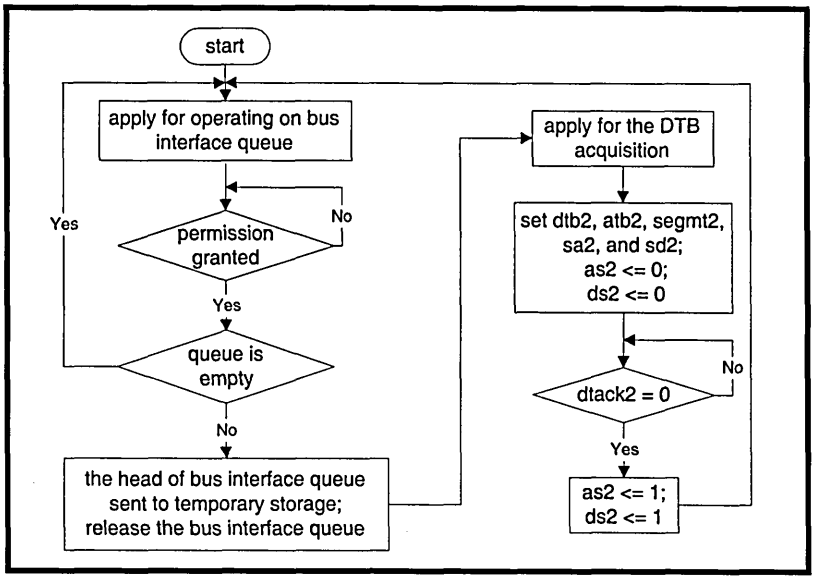


Figure 5.18 Asynchronous Channel Bus Interface (*for two buses*)

But, its implementation is more complex this time due to the cross-bus operation and the unsymmetrical gateway operations. First of all, those two queues are all critical regions. For example, the main queue is accessed by two gateways that are attached to completely independent buses. Thus this memory should not be approached simultaneously by two gateway components. Similar principle applies to another queue. Secondly, the situation three and four above-listed are entirely different from their equivalents on the same bus. They need to be taken care of with caution.

The troubles stem mainly from the read and write operations that position on different buses. The former version of surveillance in the monitor is not operative here. To contend with this problem, a separate queue is required. If the sender finds the main queue is empty and the receiver has paid a visit on the main queue (mark set to "1"), the data is obviously required to be sent directly to the receiver not to the main queue. Since the receiver is attached to another bus, its monitor, however, can not detect this data transfer on the other bus, despite having been started after the failure of receiving data from the main queue. This particular message has to be waiting in the another queue and inform the waiting receiver of its arrival. A bus interface is also required to accomplish the DTB handshaking and data transfer on bus_2. For convenience, the handshaking takes place between the bus interface and the asynchronous channel gateway_2 because receiver's monitor can now sense the DTB data transfer that is indicated in the ATB.

Having clarified the background, we can now look into individual components: asynchronous channel gateway_1, asynchronous channel gateway_2, and asynchronous channel bus interface. They correspond to Figure 5.16, Figure 5.17, and Figure 5.18.

Connected to Bus_1, gateway_1 is a read only component and responsible for admitting data from the sender. The permission to operate on main queue has to be applied for before any DTB handshaking takes place. The main queue is identified as *critical section* in Figure 5.16. Inside the critical section, the valuable *mark* is checked. A zero indicates that the receiver is not waiting for data and it can be sent into the main queue, whereas a non-zero prompts data sent into the bus interface queue to inform the receiver of arrival of the data. This queue is also a critical region because both accesses from gateway_1 and bus interface are operating concurrently.

Compared with gateway_1, gateway_2 is a write only component. Its responsibilities are:

- completing handshaking with the bus interface
- taking data from the main queue and placing them on Bus_2

In much the same way as gateway_1, it has first to apply for permission to access the main queue because of the concurrency between gateway_1 and gateway_2. In addition

to the handshaking, it checks whether the main queue is empty and set the mark to "1" if yes. This mark will be reset to "0" by gateway_1 when the data arrives from the sender. Notice the selection structure "segmt2 /= arbitra_id" in Figure 5.17. It distinguishes the handshaking request from the receiver of this asynchronous channel from a similar request but that comes from its own bus interface. The major difference in reply to these two requests is that the request from its own bus interface is a mimic handshaking. It is only used to inform the receiver of the arrival of the message, which is being observed by receiver's monitor.

The asynchronous channel bus interface illustrated in Figure 5.18 is a rather ordinary bus interface component introduced earlier, except it acquires data from the interface queue. It needs, therefore, the permit to operate on the queue thanks to the concurrent operating from the gateway_1 side. Other operations involved in the component are all related to the DTB data transfer handshaking presented already.

5.5.4 Synchronous Channels with Two Buses

This is a relatively complex implementation in comparison with other three channels. It includes two channel gateways, two bus interfaces, two interface queues, and one main memory block as the main queue. Its internal design has been shown in Figure 5.19.

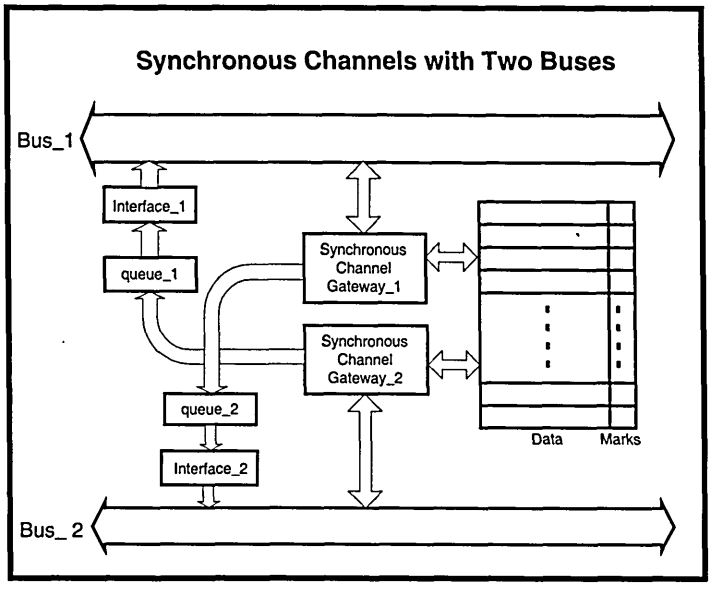


Figure 5.19 Synchronous Channels with Two Buses

Contrary to an asynchronous channel that must contain a queue in order to accommodate a series of messages, all synchronous channels across the buses can be pegged into a single global memory. Each synchronous channel has one entry and one

mark in it, which is the same structure as its fellow synchronous channel organization on same bus. It is easy to find that component gateway_1, queue_2, and interface_2 form a synchronous communication path from Bus_1 to Bus_2, while component gateway_2, queue_1, and interface_1 are amalgamated into its opposite path from Bus_2 to Bus_1. This symmetrical structure enables us only to discuss the path from Bus_1 to Bus_2 below. . In fact, the designs for queue_2, and interface_2 are same as the components in the asynchronous channel on same bus. The following content is accordingly focused on the synchronous channel gateway_1.

Figure 5.20 represents the internal design of synchronous channel gateway_1. In addition to executing DTB data transfer handshaking with its counterpart on Bus_1, this component takes care of message storage and retrieval and bookkeeping. After a component attached to Bus_1 gains control of the DTB, it contacts the gateway_1 through a synchronous channel and the channel identity is specified in the ATB. The following options must be taken into account:

1. The component is sending message and receiver is waiting (rw = 0 & remark(atb) = 1)
2. The component is sending message but receiver is unready (rw = 0 & remark(atb) = 0)
3. The component is receiving message while it is available (rw = 1 & remark(atb) = 1)
4. The component is receiving message while it is unavailable (rw = 1 & remark(atb) = 0)

It is understandable that both gateway_1 and gateway_2 concurrently access the global memory and the protection against data inconsistency should be imposed upon it. Similar principle is applied to queue_1 and queue_2.

Option one and two signify a bus-write operation mastered by the component at Bus_1. In the case of option one, i.e. the message is waited for (*remark(atb) = 1*), the control line *ready1* is set to "0" that will release the sender from waiting at Bus_1. Besides, the remark in the concerning address is set to "0" and the message is directly sent to queue_2. This message will be resent via interface_2 and acknowledged by the gateway_2. Because both of them are attached to Bus_2, the DTB data transfer can be

detected by the receiver's monitor that has bee started after it failed to receive message from the global memory earlier. In the case of option two, the receiver is not ready (*remark(atb) = 0*) and the massage is placed in the memory and the line *ready1* is assigned to "1" that results in the sender having to wait until the receiver pays a visit at the address.

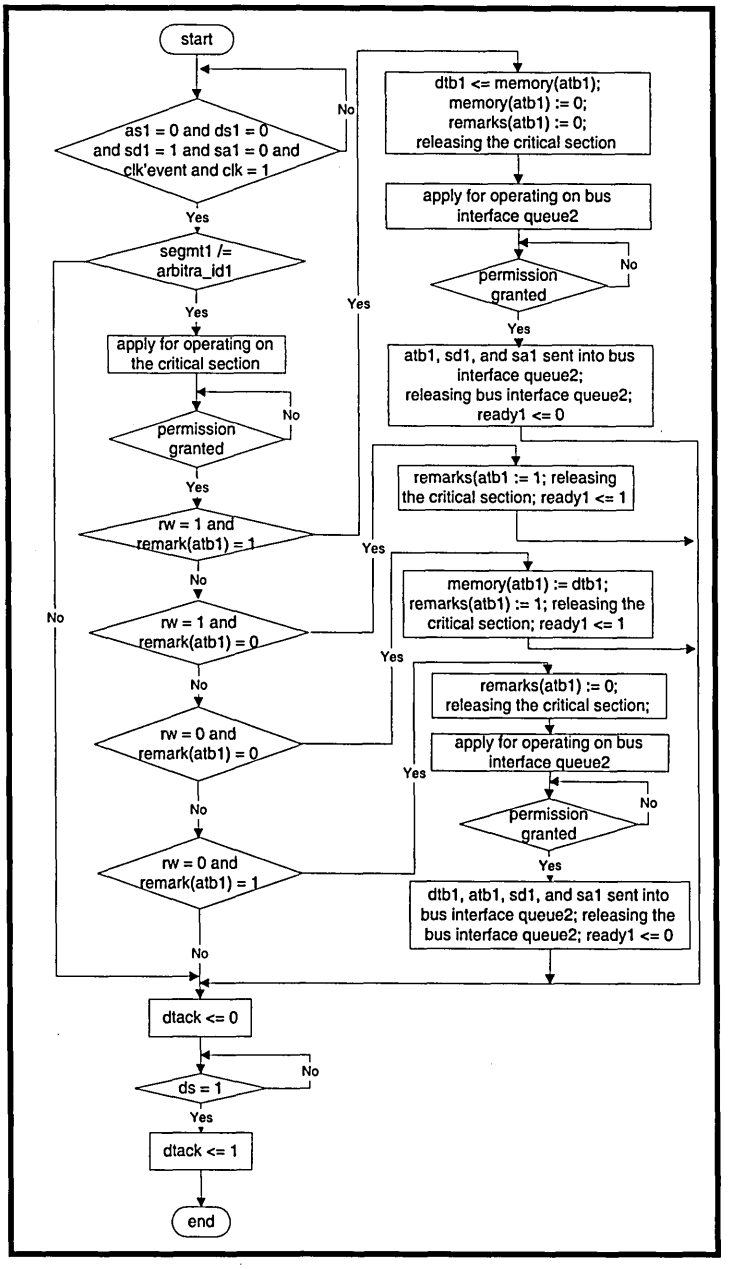


Figure 5.20 Synchronous Channel Gateway_1 (*for two buses*)

On the other hand, a bus-read operation (which *rw = 1*) is undergoing. If the message is available (*remark(atb) = 1*), the message is taken out of the memory and the remark is deleted. In addition, the sender is to be informed of the event, which has been waiting for release at Bus_2, but gateway_1 needs to apply for access to queue_2 first because

interface_2 is running concurrently with gateway_1. If the message is unavailable (*remark(atb) = 0*), both remark(atb) and ready1 are assigned to "1", which results in the sender suspending itself on receipt of the ready1 as "1".

Finally, the selection "segmt1 /= arbitra_id1" in Figure 5.20 is used to recognize whether the bus event is initiated by interface_1 or by other hardware/software component from Bus_1. While the latter position has been dealt with above, the former one is simpler. Only DTB handshaking occurs.

5.6 VHDL Packages and Libraries for Channel Communications

VHDL packages and libraries fostering the asynchronous bus protocol and communication channels have been developed in this research. In addition to VHDL packages introduced in Chapter 4, the package *ESSENTIAL_DEFINITIONS* is specially designed to support the bus protocol handling. All VHDL Components discussed earlier in this chapter have been tested and integrated into a VHDL library. It includes the following components:

- Bus_Operating_Logic
- DTB_Arbitrator
- Clock_Generator
- Synchronous_Channel_Same (*on same bus*)
- Asynchronous_Channel_Same (*on same bus*)
- Synchronous_Channel_Differ (*on two buses*)
- Asynchronous_Channel_Differ (*on two buses*)

Their VHDL source files have been included in Appendix E for inspection. Some of the components listed above contain sub-components that are written in concurrent VHDL processes. For example, while its conceptual structure has been described in section 5.4, the component: *Synchronous_Channel_Differ* is composed of the following concurrent processes plus other invocations from the VHDL packages:

- gateway_1
- gateway_2
- bus_requester_1
- bus_requester_2
- bus_interface_1

- bus_interface_2.

Thanks to the VHDL packages and libraries introduced above, VHDL co-simulation programs can be developed without the knowledge of communication protocol. The VHDL components in the libraries and packages can be readily implanted in the performance evaluation program. More important is that the VHDL packages and libraries provided in this project have been properly designed and tested. They are all generic components and can be reused across the developments of codesign applications. Furthermore, communication channels can be synthesized in much the same way as electronic device is produced from assembler line and supplied in the hardware circuitry components. The VHDL programs in such a treatment tend to be more robust, reliable and cheaper. This type of process also eliminates the need to build each application from scratch and promotes the component reuse, which is an important principle in the object-based methodology. These VHDL packages and Libraries, therefore, constitute the part of developmental contributions in this thesis.

5.7 Integrated Performance Evaluation in the Co-simulation Technique

As mentioned earlier in this chapter, the performance evaluation for codesign system relates to three factors: the performance of hardware component, the performance of software component, and the communication cost. While section 5.5 targets the last, this section caters for the other two. In addition, the integrated performance evaluation technique cemented with three factors in VHDL co-simulation program is dealt with too in this section.

5.7.1 Performance Evaluation for Software Component

Evaluating the performance of software component is relatively easy to execute due to the fact that embedded processor is normally provided with development board and software simulation tools. They facilitate the performance evaluation in emulation/simulation. This research has adopted the ARM embedded processor [Fur96] in respect that its software development toolkit is available. Consequentially, all components dispatched to software implementation during hardware/software partitioning phase are assumed to execute on ARMs and the following discussion is focused on this processor though it does not make principal difference if other processors are employed.

Our project uses the *ARM Software Development Toolkit* (SDT) supplied by the Advanced RISC Machines Ltd., which supports the C cross-development for ARM processors. It features a fully integrated development environment based on Windows 95/NT 4.0. A thorough description of the environment can be found in [ARM97]. The following three facilities are included in the SDT:

- code and data size
- overall execution time
- time spent on specific parts of an application

The performance evaluation discussed here is mainly concerned with overall execution time and the time spent on specific parts of an application whereas the code and data size are also highly beneficial to codesign, which will be exploited further in the conclusion chapter.

It is worth mentioning that we have targeted at the distributed architecture with multiple processors and the layered bus structure. Based on the single bus target architecture, previous codesign approaches allocate all software components to a single processor. These components, therefore, have to be bonded in a single C program, which is in fact a reprogramming process. Moreover, the communications between software components have to take place via parameter passing in C functions. Notwithstanding its simplification of the software communication, it restrains the design space exploration. On the other hand, the distributed target architecture increases the calibre to design space exploration by allowing multiple processors and the flexible scheme of allocation of software component. We would like, however, to leave this matter as one of the future research topics.

The ARM C compiler and the linker check the C program and then create executable code. The overall execution time and the time spent on specific parts of an application can all be obtained by using debugger facility in the SDT. As a C program executes, the SDT counts the total number of clock ticks taken, and reports this figure to user either through the debugger's $clock variable, or indirectly through a C library function such as clock(). In addition, because of debugger's symbolic feature, any program section

can be accessed against its executing clock ticks, (i.e. the time spent on this particular section), provided the following parameters are set up for the symbolic debugger:

- The type and speed of the memory attached to the processor
- The speed of the processor

An example is presented below to demonstrate the timing acquisition for a software component.

5.7.2 Software Component Performance (an example)

The C program is designed to demonstrate the time taken by *insertion sort* algorithm. The output from ARM SDT's compiler and linker is displayed in Figure 5.21 and the execution results in the ARM debugger are shown in Figure 5.22. As the Console Window indicates the *insertion sort* took 1369 clock cycles. This is obtained through C library function clock() and using processor ARM7D with clock speed 20 MHz and 2048 Mbytes on chip memory. According to the clock speed and clock cycles taken by the function *insert_sort*(), the time delay for this function is then decided, which is 450 nanoseconds. Alternatively, this can also be retained by setting *Toggle Breakpoint* in the SDT Window Toolkit and using dynamic debugging facility.

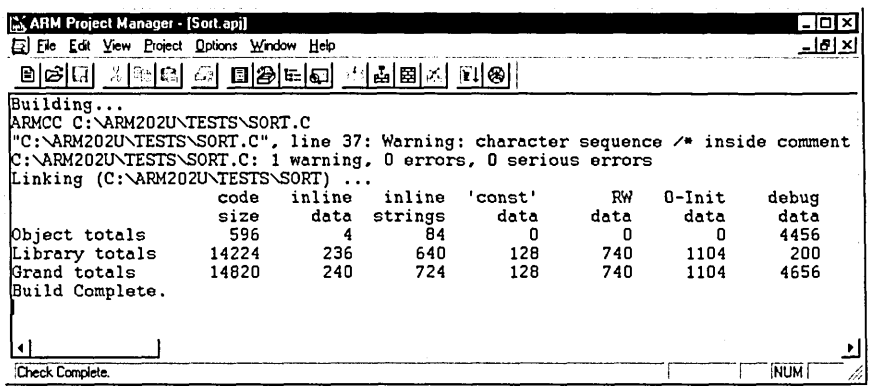


Figure 5.21 Output from the ARM Project Manager

5.7.3 Performance Evaluation for Hardware Component

The performance of hardware component is to be acquired through *High-Level Synthesis* (HLS) for chips and hardware systems. The HLS has been a well-established research area in the VLSI discipline since late 80's [Cam90]. A preliminary discussion will help appreciate the technical details described later in this section.

The following essential issues are involved in the HLS [Gaj92]:

- High-level formulation (i.e. theoretical foundations)
- Design modelling and representations
- Design quality measures
- Synthesis algorithms
- Design standardization and its supporting environments

Its research topics yet include:

- Design representation and transformations
- Partitioning
- Scheduling
- Allocation

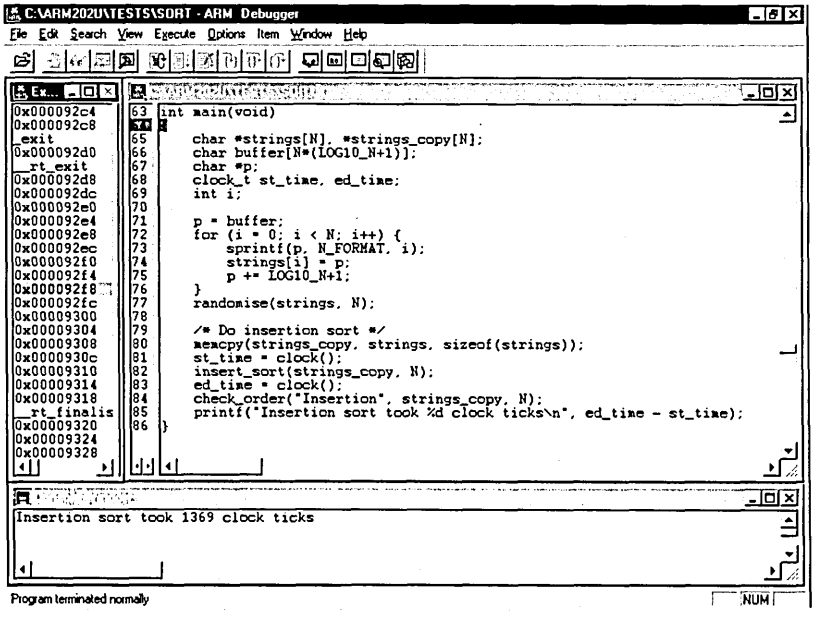


Figure 5.22 Results from the Debugger

The performance evaluation concerned here fits into *design quality measures* and *synthesis algorithms*. The former predict efficiencies of synthesized hardware design, such as manufacturing, testing and maintenance costs, silicon area, wire delay, and the performance, whereas the latter are developed to transform a high-level description into its low-level counterpart that is represented in a set of connected storage and functional units. Miscellaneous synthesis algorithms have been published [WC91] since late 80's, among which some of the algorithms have been incorporated in automatic synthesis tools. Because of variant efficiencies and emphases in those algorithms, the synthesized quality differentiates from one to another. Amongst synthesis algorithms, one particular group, *Scheduling Algorithms*, assigns each operation to a point in time that is

sometimes called a *control step*. The number of control step derived from scheduling algorithms to accomplish a task actually represents the performance of this component, which is exactly the time delay in the component when it is simulated in VHDL.

A behavioural description (the VHDL program in this project) for a hardware component specifies the operations to be performed by the synthesized hardware circuitry. During the *compilation* in hardware synthesis, the behavioural specification is converted into an internal representation, such as CDFG (Control/Data Flow Graph). It is then divided into sub-graphs, each of which is executed in one control step that accords with one state in the *Finite State Machine* (FSM) model with a Datapath. The scheduling process dispatches operations in CDFG into states or control steps. The ultimate goal of scheduling in hardware synthesis is to optimize the number of control steps required to complete a function under the constraint of hardware resource or cycle time. This research, however, does not intend to compete those published scheduling algorithms. It, instead as an integrated part of the codesign approach, incorporates those well-established algorithms in performance evaluation phase to determine the number of control step, i.e. the performance of hardware component.

Two types of general scheduling algorithm exist at present: *Time-Constrained Scheduling* and *Resource-Constrained Scheduling*. Although any scheduling algorithm is theoretically practicable to be integrated into our methodology, the *List Scheduling Algorithm* [CW91] has been chosen in this project due to its huge popularity and its capacity of trading hardware cost for its performance. Besides, we only deal with the basic scheduling algorithms with the following restrictions:

- Each operation takes just one control step.
- Each type of operation can only be performed by one type of function unit.

The List Scheduling Algorithm, outlined in Figure 5.23, is essentially a resource-constrained scheduling algorithm. In this algorithm the operation types are from t_1 through t_m. This means that the number of different operator type is m. The resource constraint for each operator type is given and reflected in the num(t_i). For any operator type t_i in the design, its priority list, P_list(t_i), is maintained. Any ready operation in the design (i.e. all its predecessors are scheduled) is arranged into its priority list and

situated at an appropriate position according to its priority. The priority function could be the mobility of an operation, i.e. the length of the longest path from the operation to another operation with no immediate successor, or the number of immediate successor. The last feature is used as the priority function in the following example.

The List Scheduling Algorithm operates on the DFG (*Data Flow Graph*) of a design, i.e. an internal representation of the behavioural description (the VHDL program in this case), which can be automatically converted by hardware synthesizer. The priority lists are initialized by the function *Initialize_Priority_List()*. Those operations without predecessors are first put into the priority lists. The operations in the lists are gradually scheduled into control steps. The number of control step is accumulated in the variable, C_Step. During the course of scheduling, other operations that were originally not in the priority lists may be added into the priority lists because of their predecessors arranged into control steps and deleted from the priority lists. This process will continue in loops until all operations of the design are scheduled.

Algorithm: List Scheduling

```
Initialize_Priority_List(V, P_list(t1), P_list(t2), ... P_list(tm));
C_Step = 0;
While((P_list(t1) /=Ø or ... or P_list(tm) /=Ø) loop
  C_Step = C_Step + 1;
  for i in 1 to m loop
    for j in 1 to num(ti) loop
      if P_list(ti) /= Ø then
        Update_Schedule(S, First(P_list(ti)), C_Step);
        P_list(ti) = Tail(P_list(ti));
      end if
    end loop
  end loop
  Update_Priority_List(V, P_list(t1), P_list(t2), ... P_list(tm));
end loop
```

Figure 5.23 Algorithm for List Scheduling

Notice that the process in Figure 5.23 operates on DFGs that is compiled from the codes with sequential structure. Other structures such as selection and repetition have to be treated with their converted CDFG (Control/Data Flow Graph) structures. We will clarify this issue in the following example.

5.7.4 Hardware Component Performance (an example)

In this example, scheduling a design in List Scheduling Algorithm and trading hardware cost for its performance are demonstrated. First, we deal with individual DFG in List

Scheduling Algorithm. It is then extended to the general cases with selection and repetition structures.

```
BEGIN PROCESS

  t_tmp:= t_tmp + 1;

  if (t_tmp < 16) then
     syn_receive(data_token, t_temp, 99999 ns);
     temp:= t_temp.color.data2;

  --/ processing 16 bits information /--
     buf_reg:= buf_reg(1 to 15) & temp;
     t_back:= syndrom(9);
     t_forward:= temp;
     syndrom:= (t_back xor t_forward) & (syndrom(0) xor t_forward) &
               syndrom(1) & (syndrom(2) xor t_back xor t_forward) &
              (syndrom(3) xor t_back xor t_forward) & (syndrom(4)
               xor t_back) & syndrom(5) & (syndrom(6) xor t_back) &
              (syndrom(7) xor t_back xor t_forward) & (syndrom(8)
               xor t_forward);

  else
  --/ sending off 16 bits information & 10 bits syndrom /--
     data_buffer(0 to 31):= buf_reg(0 to 15) & syndrom(0 to  9) & "000000";
     bus_called(clk, chanl_out, word_zero,
              data_buffer, data_temp, '0', '0', '0',
              atb_l, segmt_l, dtb_l, rw_l, sd_l, sa_l,
              enb_l, done_l);

  --/ initializing the pm_16 again /--
     t_tmp:= -1;
     syndrom:= (others => '0');
     buf_reg:= (others => '0');
  end if;

END PROCESS
```

Figure 5.24 Hardware Design in VHDL

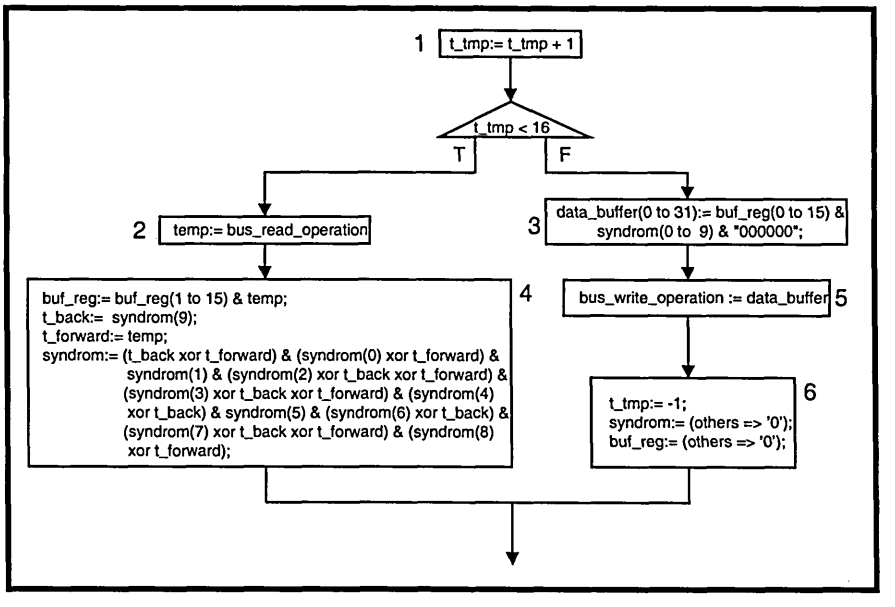


Figure 5.25 Control-Flow Structure

The original VHDL design has been shown in Figure 5.24. Its control-flow structure is illustrated in Figure 5.25. In the Control-Flow Structure, there are six rectangles

constituting sequential sections in the design. When these blocks are filled in their DFGs they produce a complete CDFG for this design.

Figure 5.26 represents the DFG converted from the Block 4 in Figure 5.25. To this DFG two algorithms demonstrated below are applied and outcomes are listed in Table 1.

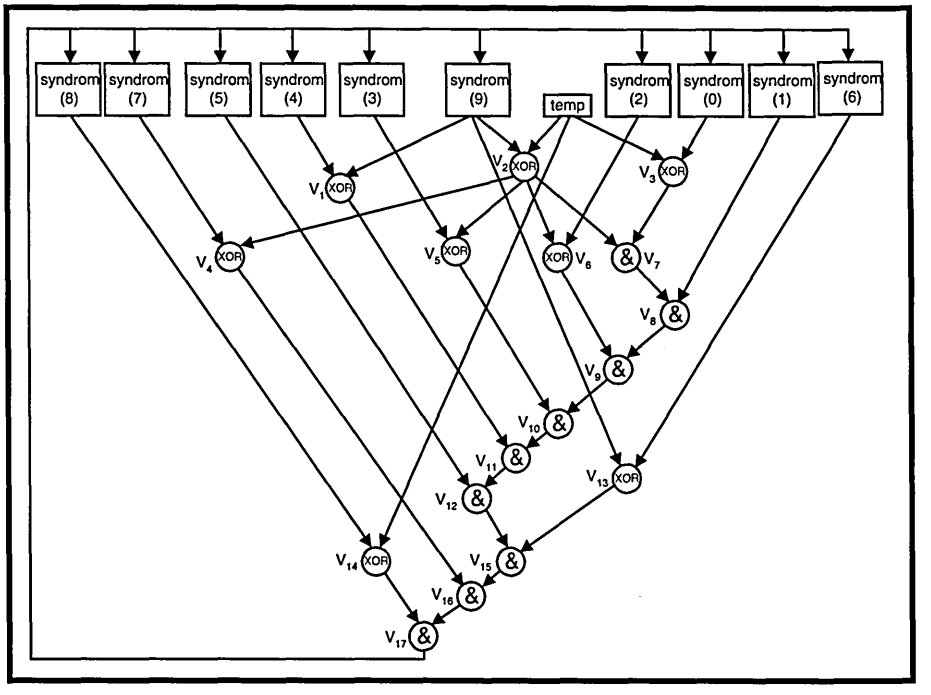


Figure 5.26 DFG for Block 4

In the table, node V_2 represents the operation "XOR". It has four immediate successors so that its priority is assigned to four. Others are all ones because they have only one immediate successor. The column "ASAP" shows the control steps, to which each operation is assigned. This particular column results from application of the ASAP (As Soon As Possible) scheduling algorithm. This fundamental scheduling algorithm, to which many scheduling algorithms refer, is exactly an ultimate List Scheduling Algorithm without resource-constraint. The List Scheduling Algorithm however requires the constraint of resource designated in advance. The ASAP scheduling algorithm here is used to find the maximum demand for hardware resource in a design and derive the reasonable resource-constraints from it. They can then be applied to the List Scheduling. For example, five "XOR" operations and one "&" operation are required in the ASAP algorithm, which implicates four other combinations as possible resource-constraints. From Schd_1 through Schd_4 forms individual scheduling scheme in the List Scheduling Algorithm.

Table 1 includes both hardware costs (*Components*) and their corresponding performances (*Total Steps*). As a byproduct, this can be used to trade hardware cost for its performance for individual hardware component although it is not the focus of this project. It is clear that Schd_2 is the most cost-effective scheduling scheme because it reaches the highest execution speed, i.e. 10 control steps, and at the same time the least hardware cost, compared with ASAP and Schd_1.

Table 1.

Node	Priority	ASAP	Schd_1	Schd_2	Schd_3	Schd_4
V1	1	1	1	1	1	2
V2	4	1	1	1	1	1
V3	1	1	1	1	2	3
V4	1	2	2	2	3	6
V5	1	2	2	3	4	7
V6	1	2	2	3	4	8
V7	1	2	2	2	3	4
V8	1	3	3	3	4	5
V9	1	4	4	4	5	9
V10	1	5	5	5	6	10
V11	1	6	6	6	7	11
V12	1	7	7	7	8	12
V13	1	1	1	2	2	4
V14	1	1	2	2	3	5
V15	1	8	8	8	9	13
V16	1	9	9	9	10	14
V17	1	10	10	10	11	15
Total Steps	N/A	**10**	**10**	**10**	**11**	**15**
Compo-nents	N/A	5 "XOR" 1 "&"	4 "XOR" 1 "&"	3 "XOR" 1 "&"	2 "XOR" 1 "&"	1 "XOR" 1 "&"

We have to extend the scheduling algorithm to take on general behavioural descriptions with selection and/or repetition structures. The CDFG structures, as illustrated in Figure 5.25, will be used in the following discussion.

A selection structure corresponds to "**if**", "**if else**" or "**case**" statement in the behavioural description. Potential control-flow structures are drawn in Figure 2.27. As demonstrated earlier, the List Scheduling Algorithm can help determine the control steps and costs for individual sequential blocks such as A, B, A_1, through A_m in Figure 2.27. Additionally, a scheduling algorithm has to allocate operators for evaluation of the conditional expression and comparison. It is noticeable that although there may be multiple threads inside a selection body, such as A_1 through A_m in a *case* statement, the operators in a selection structure can be shared economically across the threads, for they are mutually exclusive depending on the outcome of evaluating the conditional

expression. Bearing this in mind, we can treat a selection merely as a normal sequential block. The only difference is that the total operator required in a selection is the union of individual thread's. This scheduling policy can be further applied to the CDFG in Figure 2.25. Two extra control steps need to complete operation "t_tmp + 1" and comparison "t_tmp < 16". The hardware cost is the union of two threads' costs that are five "XOR"s and two "&"s, plus one adder and one comparator.

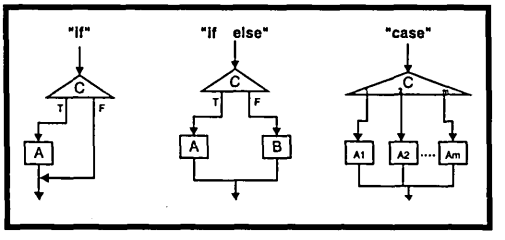


Figure 5.27 Control-Flow Structures

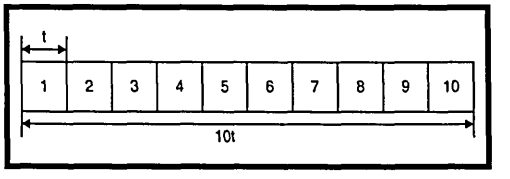


Figure 5.28 Loop Scheduling

The performance of a selection, however, can not determined in scheduling phase since the conditional expression is dynamically evaluated during simulation. This difficulty does not pose a serious problem. It can be solved in co-simulation explained in the next section.

On the other hand, a repetition structure corresponds to **"while"**, **"for"** or **"loop"** statement in the behaviour description. A scheduling algorithm can only practically schedule a loop with definite repetition time, such as **for** statement in VHDL. Components with other repetition structures are deemed to implement in software. There are three different ways of scheduling a loop [Gaj92]: *Sequential Execution*, *Partial Loop Unrolling*, and *Loop Folding*. The first method is adopted in this project. We assume that the loop has n iterations and each loop needs t control steps and hardware cost c to complete. The total execution time is therefore nt. An example is shown in Figure 2.28. Suppose the loop has 10 iterations and each needs t control steps to finish. The total execution time is therefore 10t. The hardware cost is the cost in repetition body.

It should be pointed out that the scheduling algorithm in this approach is not aimed at synthesizing a hardware design into circuitry, which can well be accomplished by commercial hardware synthesizers. The method introduced here is only used to

determine the hardware cost and the execution time wherever possible. Other unsolved performance issues will be dealt with in the system co-simulation phase.

5.8 Performance Evaluation for Codesign System

We have so far furnished the methods purely to evaluate individual performances of hardware/software components and the communications across the designated system target architecture. These individual methods will be pieced together in this section to create an integrated approach to performance evaluation for codesign system, which provides a flexible approach to performance evaluation and is also the strength of our methodology.

VHDL simulation (co-simulation here) plays a vital role in our approach. Compared with previously published research, we pursued a hardware-based co-simulation approach, i.e. the co-simulation supported by the VHDL simulation environment. This route has been pursued because of the following reasons. First of all, VHDL language supports hardware design. Its simulator provides the performance evaluation for hardware components. Secondly, thanks to the VHDL synthesis tools, most of the VHDL design can be automatically synthesized into hardware circuitry, which speedups the time to market and improves design quality of hardware component.

VHDL supports three abstraction models: a behavioural model, a timing model, and a structure model [Ash96]. Here, first two models are involved to support the co-simulation. A *behaviour* is a functional interpretation of a hardware system, while the timing in hardware design designates the amount of time elapsed during simulation. Simulating a design in VHDL concerns two separate issues: functionality and timing. The first one is same as in software execution at the behavioural level, while the second is unique in hardware simulation. The major difference between hardware and software design is that a hardware designer has to handle the timing in a circuitry whereas the timing in software design is handled by the operating system. Utilizing this feature properly, however, one can fully manipulate the timing with regard to hardware/software components and the communication between them, so that the evaluation of the system performance can be achieved.

As aforementioned, while a behavioural design is evaluated for its functionality, it makes no difference whether it is simulated by hardware simulator or executed by operating system. This rationalizes the practice, in which a VHDL simulator undertakes evaluations of functionality for both hardware and software components. The performance of individual hardware/software component can be represented by insertions of time delay statements (*annotations*) in the relevant places in VHDL the program. In addition to those statements accomplishing the evaluation of functionality, a VHDL co-simulation program thus includes timing annotations (time delays) for individual hardware/software components and the time delay for communications that is supported by the VHDL packages and libraries developed in this project. The annotations for hardware component come from the result of scheduling process and the time delay for software component is based on the clock cycles that are retained during symbolic debugging in the ARM's SDT. Since the simulation is positioned at the higher level, it avoids the problems of simulation efficiency as mentioned in Section 5.1 and, at the same time the performance evaluated in the co-simulation could be accurate to system clock cycles.

The case study presented in Chapter 6 will demonstrate how the codesign approach proposed in our research is applied to solving a codesign problem. The technical details described in this chapter will become clear when they are applied to a comprehensive case study.

5.9 Review

In terms of system target architecture previous research was focused on the conventional target architecture, which is composed of a single processor, single bus, and hardware component(s). The cosynthesis method involving this architecture is rather straightforward, in which all software components are confined in the processor and the communications between software components are simplified in the parameter passing between procedures or/and functions. The communications between hardware and software components are implemented in bus communications via an intermediate global memory or system interruption. The evaluation of the system performance has to be tied to this inflexible target architecture.

In order to increase the system throughput, current practice in codesign discipline is to dispatch those computationally intensified components to hardware implementations. But, those component does not always exist in codesign system. This causes the problem in which hardware implementations do not necessarily improve the system performance. It results from the fact that if the system communication dominates over the computation load of individual component there is little effect on the system performance by dispatching more system components to hardware implementation. The research results published in [Edw97] support this point of view.

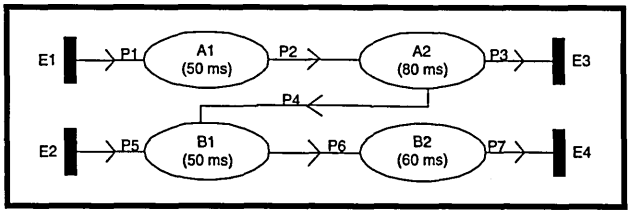


Figure 5.29 System Process Graph

Our approach, however, goes beyond the conventional single bus system target architecture. The methodology we developed allows system target architecture to be exploited for higher communication throughput. This feature is supported by the layered system bus structure described in this chapter. An example of exploration of bus layer to improve the system communication throughput is shown in Figure 5.29 and Figure 5.30.

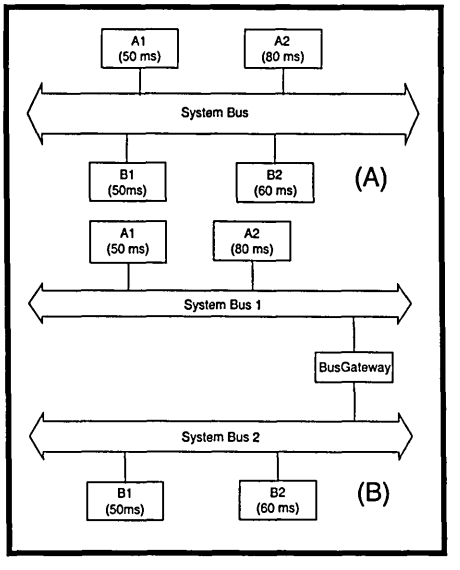


Figure 5.30 Exploration of System Target Architecture

In this example, we assume that the system being codesigned comprises four functional components A1, A2, B1, and B2. They have similar computation loads as indicated inside each function. External entities E1 and E2 provide raw data for processing in the system and E3 and E4 store the processed data. The synchronous communication paths, P1 through P7, are supposed to dominate over system's computation. We also assume that the communication loads in paths P1 through P3 are higher than the loads in paths P5 through P7. Besides, the communication load in P4 is much light than the loads in other paths.

Although in the partitioning phase these four components can be all dispatched to hardware implementations, the system performance does not significantly improve if they are connected to the same system bus as shown in Figure 5.30 (A). This is because the waiting time for each component to access to the system bus now overtakes the execution time of individual component.

The system target architecture supported in our methodology is however flexible, compared with the conventional architecture. Additional system bus can be added if the communication throughput should increase. In Figure 5.30 (B), the system target architecture with two bus layers is employed to increase the system communication throughput. Furthermore, the allocation of components A1, A2, B1, and B2 within these two buses also exerts impact on system's performance. Because the communication loads in paths P1 through P3 are higher than the loads in paths P5 through P7, A1, A2, E1, and E3 are assigned to the *System bus 1* as shown in (B), which has higher throughput. The components B1, B2, E2, and E4 are connected to the *System Bus 2*. This allocation scheme can produce better system performance due to the second system bus introduced to promote the system's communication throughput.

While this example is dealt with in the principles introduced above, the next chapter will use a case study to demonstrate the quantitative impact on system performance in relation to the varied component allocation schemes.

Major contributions from this chapter are summarized as follows:

- The asynchronous bus protocol has been designed, based upon which the layered system bus structure has been created. In addition the VHDL packages supporting the virtual prototyping and co-simulation in VHDL programs are produced.
- The integration of the *ARM SDT Tool Kit* and the *List Scheduling algorithm* into the proposed methodology and the cosynthesis method in which the system components are allocated to the layered system bus structure have also been devised, which together with the dispatch of system components towards hardware implementations improves performances of both individual components and the whole system.

Chapter 6

Case Study

The feasibility and strength of our Co-PARSE Codesign Methodology is evaluated by a case study: *Codesign of RDCS (Radio Data Computing System)*. The *RDCS* is a new Radio Data Application System. It extends current RDS (*Radio Data System*) technology by introducing the *parallel channel processing* and the *data computing* techniques. The design of the RDCS has been supported by our codesign approach. Tasks undertaken in this study are listed below:

- Co-specification of RDCS in Enhanced Process Graph and Co-BSL program
- Functional verification and system profiling in VHDL simulations
- Hardware/software partitioning and component allocation
- Performance evaluation for software components in ARM SDT
- Performance evaluation for hardware components in Listing Scheduling Algorithm
- Co-synthesis of interfaces of hardware/software with virtual prototyping technique
- System performance evaluation in VHDL co-simulations
- Analysis of the co-simulation results

6.1 Fundamentals of RDS

Since the RDC system is an extension of the RDS that is the cornerstone in this case study, a general introduction to the RDS is first given to assist in understanding the design details. Followed are the details of the RDCS itself.

The RDS [MW90] [RDS98] was originally designed to broadcast digitally encoded information over Europe, including UK in late 80's. It was especially intended to assist the car receivers in the task of proper tuning. Because of the increasing demand for information and the dramatic technology advance, its potential use has well been beyond its original target. The European Standard EN 50067 has become the definitive standard in Europe and the UK [Bri92]. It has been designed to be upwardly compatible allowing the broadcaster and receiver manufacturer to add or to extend existing features when the demand develops. This case study develops a new feature of RDS, which is the Radio Data Computing.

The RDS technology was established on the fact that modulation bandwidth of frequency modulated transmitter is typically about 90 kHz but the required bandwidth for stereo sound signals is some 53 kHz, which provides an extra radio resource carrying additional information in current analogue radio broadcasting channels. The data signals are carried in a sub-carrier that is added to the normal VHF/FM radio signal at the input to the ordinary VHF/FM transmitter.

The RDS broadcasting has been illustrated in Figure 6.1. Radio data signals are fed into the radio data encoder. Encoded data signals are carried in a subcarrier, which is amplitude-modulated by the shaped and biphase coded data signals, and then added to the stereo multiplex signal inside the VHF/FM transmitter. After the radio (and data) signals are demodulated in the VHF/FM receiver and prior to any de-emphasis, the radio data signals are separated from the multiplexed signal by a sharp band-pass filter and then demodulated and differentially decoded. Further data processing such as synchronization or error correction takes place in the radio data decoder.

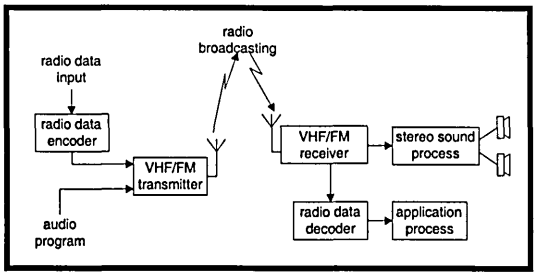


Figure 6.1 Scenario of RDS Broadcasting

This case study is not concerned with radio broadcasting that is detailed in the physical layer [Spe84]. It instead deals with data decoding and application, i.e. upwards from the data-link layer. In data-link layer, the encoded data signals are organized in a 16-bit word followed by a special 10-bit CRC error correction checkword [PW72], which forms a 26-bit block. Four consecutive blocks produce a group that is the biggest unit in the baseband structure as shown in Figure 6.2.

The radio data decoder in Figure 6.1 receives binary bits from the VHF/FM receiver and performs error protection. The principle of error protection is as follows. Each transmitted 26-bit block contains a 10-bit checkword that is primarily used by the radio

data decoder to detect and correct errors that occur during transmission. The checkword is the sum of the following:

- The remainder after multiplication by 10^x and then division (modulo 2) by the generator polynomial g(x), of the 16-bit information word
- A 10-bit binary string d(x), called the *offset word*

Where the g(x) is given by:

$g(x) = x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$

and where the offset values, d(x) is different for each block within a group. They are listed in Table 1.

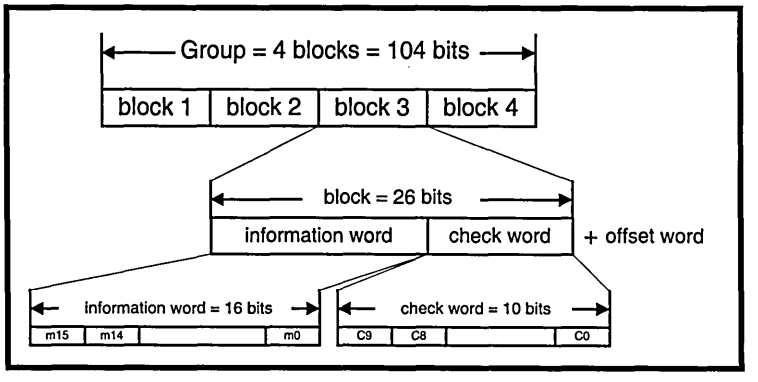


Figure 6.2 RDS Baseband Coding Scheme

An obvious difficulty is how effectively to implement the multiplication and division of the polynomials introduced above. There are several methods published for this purpose [PW72]. They are either hardware or software based technique. This study employs the most popular approach based on the shift-register arrangement i.e. hardware based method. An informative documentation about this technique can be found in [Spe84].

Table 1.

Offset	***Offset Word***
Block1	0011111100
Block2	0110011000
Block3	0101101000
block4	0110110100

The session and presentation layers specify the message format and addressing structure in the RDS. The message and addressing structure in a group is described in Figure 6.3. Those relevant abbreviations are listed as follows: *PI* = Program Identification, *PTY* = Program Type Code, B_o = Version Code, *TP* = Traffic Program (identification code) and *Check word + offset* (10 bits). The purpose of offset word is to provide a group and

block synchronization mechanism in the receiver/decoder. The added offset is reversible in the decoder and the normal additive error correcting and detecting properties of the basic code are unaffected. All information or check words, binary numbers or binary address values have their most significant bit transmitted first. There is no gap between the groups or blocks.

Although the message format could be considerably complex by combination of binary bits representing *PI*, *PTY*, B_o, and *TP*, it is simplified here because this study is not designed to tackle other application issues. In addition, it is impractical to simulate inclusively RDS applications due to huge quantity of special stimuli required. As a result, several reasonable simplifications have been made.

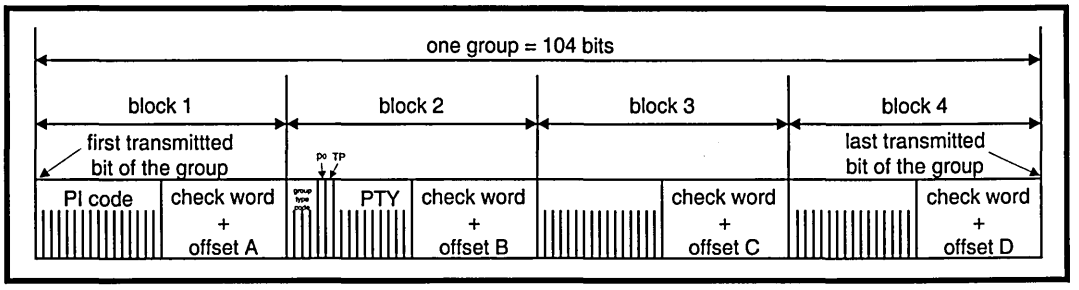


Figure 6.3 Messaging and Addressing Structure of a Group

The data signals are designated in two types: numbers and text (ASCII code). The addressing structure is as follows:

- The PI Code of Block 1 in every group is presented in the binary code "1100000000000000".
- The Group type Code is "1111" for numbers and "0000" for text in Block 3 and 4.
- The PTY, B_o, and TP are all set to 0s.

6.2 Radio Data Computing System

The RDS is designed to encompass variety of new applications. An experimental application is charted in this project that is named *Radio Data Computing* (RDC). It is designed to receive integers concurrently from two separate RDS transmitters (stations) and perform designated numerical calculations. This new application enables us to demonstrate the communications via the target architecture with layered bus system aforementioned in Chapter 5. Particularly, we assume that the numerical calculation is as follows:

$$C = F_a(A) * F_b(B)$$

Where functions F_a and F_b are applied to matrix A and matrix B respectively. The matrix resulted from $F_a(A)$ is then multiplied by the matrix $F_b(B)$ and the product is sent to matrix C. Elements of two matrices A and B are encoded and transmitted from station A and B separately.

The RDC system is codesigned, in the codesign approach developed in this project. The major codesign issues are addressed and the experimental results are displayed and explained.

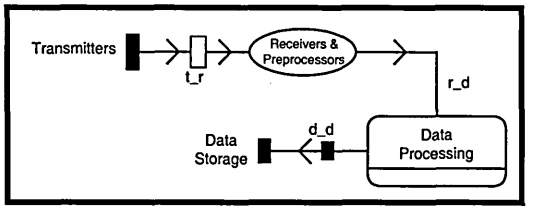


Figure 6.4. Top-Level Process Graph

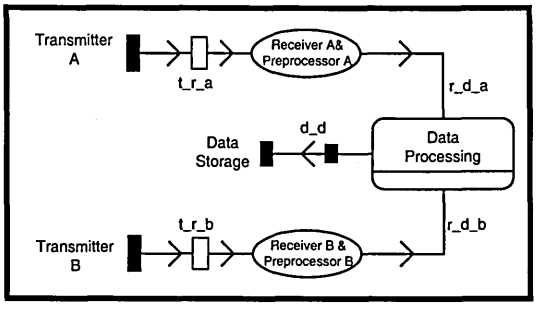


Figure 6.5. Second-Level Process Graph

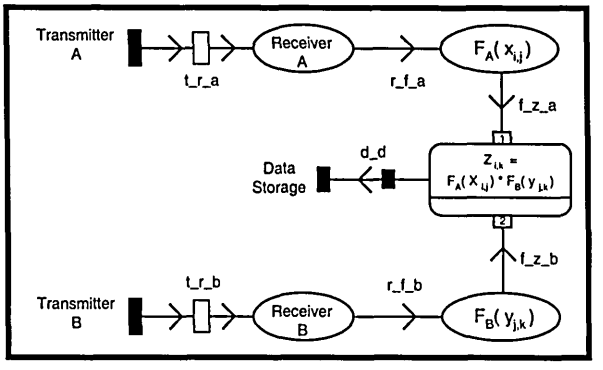


Figure 6.6. Third-Level Process Graph

6.2.1 Co-specification of RDCS in Enhanced Process Graph

The RDC system has been specified in the Extended PARSE Process Graph, and then gradually been refined as shown in Figure 6.4 through Figure 6.7. At the top level (Figure 6.4), a function server (*Receivers & Preprocessors*) and a control process (*Data Processing*) represent the RDS decoders and application processes as illustrated in Figure 6.1. All other irrelevant details are abstracted in two external entities, *Transmitters* and *Data Storage*. The results from the Data Processing are preserved in the external entity *Data Storage* through an asynchronous channel. A *wire* path is used because the transmitters emit radio message regardless of the readiness of receivers.

The Co-BSL specifications in relation to the top-level process graph and the low-level process graph are included in Appendix F, which forms the basis for the refinement that subsequently results in the VHDL program shown in Appendix J.

The low-level process graph is depicted in Figure 6.7, in which two decoders and two numerical processors are employed concurrently to process two data streams from separate RDS transmitters. Function server PM_16_A/B processes the first 16-bit information word, while the next 10-bit checkword is dealt with by PCW_10_A/B. In addition, function server Corrector A/B executes the error check/correction, whereas function servers $F_A(X_{i,j})$ and $F_B(X_{j,k})$ apply the functions F_a and F_b to elements in matrices A and B. The matrix multiplication takes place in the control process Control Matrix. Apart from wire paths and an asynchronous path, other inter-process communications are all synchronous channels because of the waterfall style in data process.

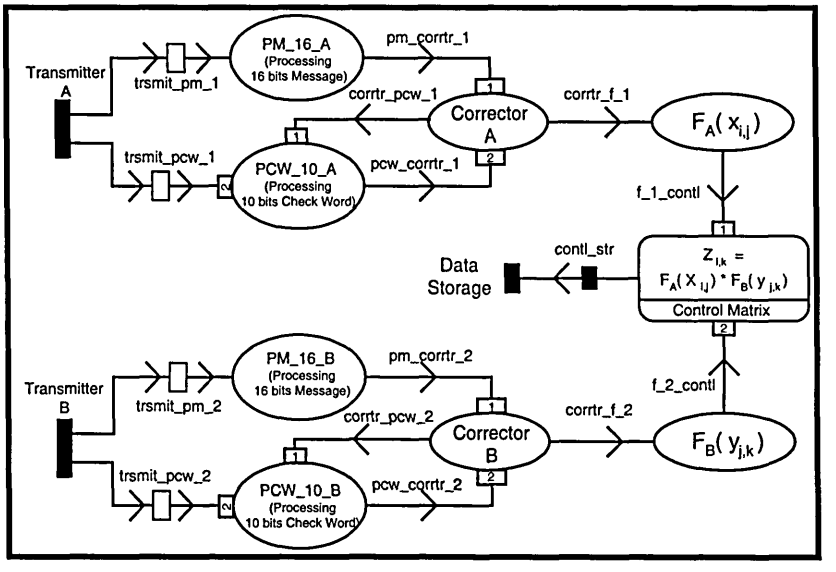


Figure 6.7. Low-level Process Graph

It is worth noticing that the specification here is neither hardware nor software biased because the processes in the process graph are neutral and they could be dispatched to either hardware or software implementation. In addition, the components in the design are being mapped to their hardware/software counterparts with whole structures retained. Therefore they can be reused in other codesign projects. It also supports object-based properties such as encapsulation and abstraction.

6.2.2 Functional Verification and System Profiling (stage 1)

The work in this section corresponds to the stage 1 of our proposed codesign framework as shown in Figure 2.6. The embedded system with time restraint compels that the system must both work properly and perform within time constraints. However, these two issues must be tackled separately. First, the VHDL program (V_1) is created for system functional verification plus system profiling. It is converted from its Co-BSL description (Appendix F) according to guidelines exhibited in Chapter 3. The major body of the program has been attached to Appendix G for reference. Second, the previously created program V_1 is converted to the VHDL co-simulation program (V_2) for system performance evaluation, which will be discussed when the system performance is evaluated later on. The major differences between these two programs lie in the following:

- While V_1 is supported by the token passing protocol as discussed in Chapter 3, V_2 is primarily based upon the layered bus protocol designed in Chapter 5.
- V_1 is used for verification of system functionality, independently of system performance, whereas V_2 has to guarantee both functional correctness and satisfactory performance.
- Extra VHDL statements attached to V_1 are those for system profiling (the usage of individual component and the communicating overhead for communication channel), but V_2 is attached by those only for time delays (annotations).

Two matrices, A_{10x10} and B_{10x5}, are generated as simulation stimuli. The resultant matrix C_{10x5} from simulation proves that the correct system function has been achieved. Besides, extra VHDL codes have created profiling information listed in Table 2 and 3. Table 2 illustrates the channel communication loads whereas Table 3 indicates the invocation time for individual process (component).

6.2.3 Hardware/software Partitioning and Component Allocation (stage 2)

The work in this section corresponds to the stage 2 of our proposed codesign framework as shown in Figure 2.6. During the hardware/software-partitioning phase, processes in the system are dispatched to either hardware or software implementation in accordance with system criteria, which have been designated as both system performance and

hardware cost in this case study. In other words, the final codesign system shall be a system with high performance and low hardware cost.

Table 2.

Channel	Communication
pm_corrtr_1	3.00 KB
corrtr_pcw_1	1.19 KB
pcw_corrtr_1	1.19 KB
corrtr_f_1	1.79 KB
f_1_contl	1.77 KB
contl_str	900 Bytes
pm_corrtr_2	1.54 KB
corrtr_pcw_2	624 Bytes
pcw_corrtr_2	624 Bytes
corrtr_f_2	936 Bytes
f_2_contl	918 Bytes

Table 3.

Comp. Name	Invocation
PM_16_A	2.66 KB
PCW_10_A	1.69 KB
Corrector A	141 Bytes
FA(Xi,j)	141 Bytes
PM_16_B	1.37 KB
PCW_10_B	893 Bytes
Corrector B	47 Bytes
FB(Xj,k)	47 Bytes

Thanks to the new capacity for the system target architecture to be exploited in our methodology, the design space exploration is now extended beyond the traditional hardware/software partitioning. A new dimension has been established, which is the *component allocation*. It decides where a component is dispatched. In addition to being sent to hardware/software implementation, a component can be allocated to a specific bus layer in order to achieve high system performance. To demonstrate this merit, the system target architectures with up to two bus layers have been trailed in this study and each target architecture has been allocated two processors.

As stated in Chapter 4, this research is not aimed at automatic partitioning process. Instead a heuristic approach has been adopted to facilitate partitioning and allocation. First of all, the process Control Matrix has to be dispatched to software implementation because of its nested repetition structure that is naturally suitable for software implementation. Other considerations are discussed in the interface co-synthesis. Second, by the heuristic conveyed in Table 3, the processes PM_16_A/B and PCW_10_A/B can be regarded as candidates for hardware implementations because of their invocation frequency. Moreover, the heuristic conveyed in Table 2 indicates that the channels connected to Transmitter A are much busier than channels connected to Transmitter B. This indicates that an allocation scheme assigning those channels connected to Transmitter A to a designated system bus could beat the communication overhead.

Bearing in mind the partitioning schemes above, we will experiment with two allocation schemes. The one is to connect all components to the single system bus in conjunction with the target architecture with only one bus layer. The other is to assign process PM_16_A, PCW_10_A, Correct A, and F_A to Bus A and the rest to bus B. Besides, alongside the partitioning schemes aforementioned, a much wide range of partitioning schemes will be tested in co-simulation, in order to verify the hypotheses made above.

Table 4.

No.	PM_16_A	PCW_10_A	Corrector_A	$F_{\_A}$	PM_16_B	PCW_10_B	Corrector_B	$F_{\_B}$
1	H	H	H	S	H	H	H	H
2	H	H	H	S	H	H	H	S
3	H	H	H	S	H	H	S	S
4	H	H	H	S	H	S	S	S
5	H	H	H	S	S	H	S	S
6	H	H	H	S	S	S	S	S
7	H	H	S	S	H	H	H	H
8	H	H	S	S	H	H	H	S
9	H	H	S	S	H	H	S	S
10	H	H	S	S	H	S	S	S
11	H	H	S	S	S	H	S	S
12	H	H	S	S	S	S	S	S
13	H	S	S	S	H	H	H	H
14	H	S	S	S	H	H	H	S
15	H	S	S	S	H	H	S	S
16	H	S	S	S	H	S	S	S
17	H	S	S	S	S	H	S	S
18	H	S	S	S	S	S	S	S
19	S	H	S	S	H	H	H	H
20	S	H	S	S	H	H	H	S
21	S	H	S	S	H	H	S	S
22	S	H	S	S	H	S	S	S
23	S	H	S	S	S	H	S	S
24	S	H	S	S	S	S	S	S
25	S	S	S	S	H	H	H	H
26	S	S	S	S	H	H	H	S
27	S	S	S	S	H	H	S	S
28	S	S	S	S	H	S	S	S
29	S	S	S	S	S	H	S	S
30	S	S	S	S	S	S	S	S

A careful consideration is here demanded. Since Control Matrix has been dispatched to a processor, say processor B, sensible process dispatch can only follow the pattern with which software implementations stretch gradually outwards, starting from the process Control Matrix. Practical hardware/software dispatch schemes are illustrated in Table 4. This necessity is also reflected in Figure 6.7, in which there is only one communication channel connected to Control Matrix from each RDS channel. Any partitioning scheme to dispatch two processes without direct connection to a same processor will result in the regrouping of processes during the implementation stage, which fails to comply with the principle of object-based development and the component may not be reused.

Besides, in the system target architectures and the processors discussed above, Table 4 forms all possible combinations of hardware/software partitions. The performance evaluation phase will examine the system performance and the hardware cost for each row in the table.

It is not difficult to understand that there is no clear cut between partitioning, allocation, and interface co-synthesis. A definite partitioning and allocation scheme will certainly affect the interface design at the interface co-synthesis stage, which will be mentioned when the interface synthesized.

6.2.4 Performance Evaluation for Software Component (stage 5)

The work in this section corresponds to the stage 5 of our proposed codesign framework as shown in Figure 2.6. As shown in Chapter 5, the ARM SDT has been integrated into our codesign methodology assisting the assessment of software performance. But, this first requires a C program to be converted from its Co-BSL description, which has been presented in Chapter 3. In fact, this C program is a mirror program of the VHDL program for functional verification introduced earlier. The C program source file has been included in Appendix H.

In the C program, compared with its VHDL counterpart, differences exist but they are reasonably allowed to remain. For example, VHDL's process is used to implement the *process* (components) specified in the Process Graph, whereas these *processes* are embodied in functions/procedures in C program. In addition, the communication between VHDL processes is by means of signal while it is via parameter passing in C program. These differences do not have much significance here because a VHDL process is simulated sequentially same as a C function is executed. The executable statements in a process/function body dominate the performance of that component (process/function). The ARM SDT debugger undertakes examinations for both the performance of executable statements and the time spent on parameter passing.

On the whole, there are three types of communication interfaces in a codesign system:

1. software vs. software component
2. hardware vs. software component

3. hardware vs. hardware component

Type 2 and 3 relate to the communication via system bus(es) that will be discussed in the *interface co-synthesis* in section 6.2.6. The problem with type 1 is associated with software allocation policy. If all software components are dispatched to one processor, the interface among them are simple, i.e. all communications are parameter passing. In this project, however, the system target architecture is being exploited and the processor in target architecture is addible. This difference results in a situation in which some of the software interface could end up in bus communication if the sender and receiver are allocated to different processors. The performance of software component, therefore, should be cautiously assessed together with its interface.

Table 5.

Comp. Name	EXEC. TIME
PM_16_A/B	483 us
PCW_10_A/B	316 us
Corrector A/B	136 us
$F_{A/B}$	6 us

The result from ARM SDT debugger is shown in Table 5. Notice that although the ARM debugger can theoretically examine the execution time accurate to individual assembly instruction, the debugger we used in this research is a freeware version that can not provide the precision up to a nanosecond. This means that the meaningful examination has to be based on the whole function body instead of the individual instruction as discussed in Chapter 5. Consequently, the accuracy of the co-simulation is slightly compromised in terms of execution time. This treatment nevertheless does not pose serious problem because the objective of co-simulation is to determine the impact on system performance under various partitioning schemes. The most important attribute here is the fidelity i.e. the percentage of correctly predicted comparisons between different partitioning schemes. The fidelity is obviously not compromised.

6.2.5 Performance Evaluation for Hardware Component (stage 3)

The work in this section corresponds to the stage 3 of our proposed codesign framework as shown in Figure 2.6. As discussed in Chapter 5, the performance of hardware component is determined in the following procedure. First the Co-BSL specification for individual component is converted to its equivalent VHDL entity description. The entity's behaviour body is a VHDL process. Next, CDFG and DFGs are constructed from this behaviour description. Since the List Scheduling Algorithm was originally

designed to be applied to sequential parts in the behaviour description, an extended List Scheduling Algorithm has to be employed to decide the performance of the whole component. Different from the approach applied to software component, the performance of hardware component is manually evaluated in our codesign approach and its precision does not rely on the particular tool available so that the hardware performance evaluated is more accurate than its counterpart in software components.

Appendix I presents the documentation resulted from the operations described above. It is organized in the following fashion. There is an entry for each major component in the RDC system. In each entry, a VHDL entity is first shown, followed by its CDFG and main DFGs. Finally a table shows the performances and hardware costs under the different hardware resource limits. These results come from the extended List Scheduling Algorithm, which is applied to the CDFG and DFGs. Two external entities and the control process Control Matrix are not supposed to be implemented in hardware component. There is no entry for them.

It is noticeable that each table indicates miscellaneous hardware costs under various resource constraints and their corresponding performances determined by the List Scheduling Algorithm. In system performance evaluation phase we are mainly concerned with the hardware cost in a codesign system with which its components have been dispatched to hardware/software implementations. Only one resource constraint that is ASAP is used during the system performance evaluation. Other costs listed in tables provide a variety of choices. They are left over to the further research topic.

The actual performance is dynamically determined during co-simulations.

6.2.6 Co-synthesis for Interfaces of Hardware/Software (stage 4)

The work in this section corresponds to the stage 4 of our proposed codesign framework as shown in Figure 2.6. Compared with traditional codesign approaches, the interface co-synthesis in our methodology is more versatile due to the flexible communication path available. As mentioned earlier, the interface co-synthesis closely relates to partitioning and allocation schemes, so that a codesign system under various partitioning/allocation schemes could have different interface designs. The following

discussion is built upon the previous assumptions, in which the system target architecture has up to two bus layers and two processors are involved. All components except Control Matrix that has been dispatched to software implementation have been listed in Table 4 together with their implementation indications.

Analyzing the alternatives in Table 4, we can find that an interface in the codesign system can be one of the following:

1. Software vs. software
2. Hardware vs. hardware
3. Hardware vs. software

The first case has been readily taken care by the parameter passing in C program during the performance evaluation for software component. The last two interfaces are synthesized in bus communication via the layered bus system. Required by the asynchronous bus protocol developed in Chapter 5, each process has been given a binary identification number for bus contention/arbitration and each communication channel is also given a binary number for bus addressing. Figure 6.8 illustrates those allocations in 16-bit binary numbers.

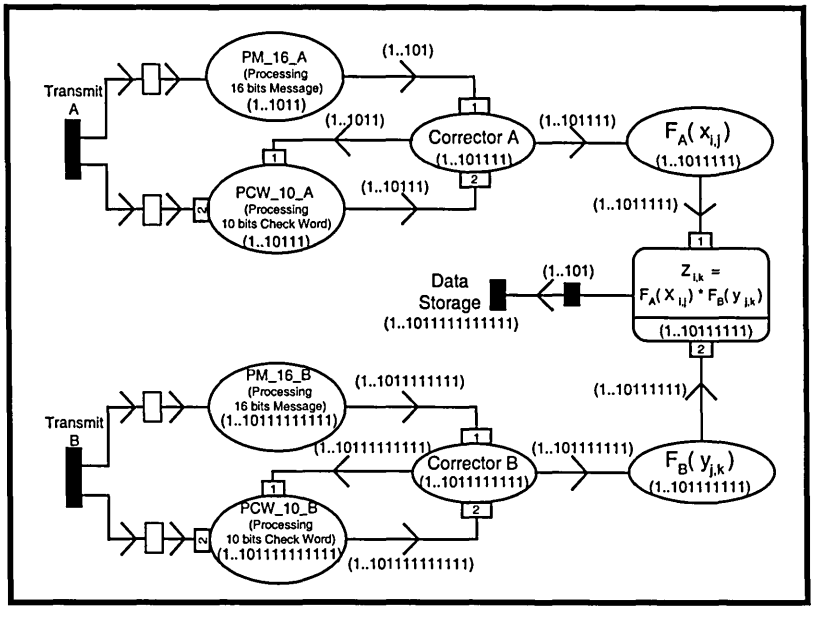


Figure 6.8. Process Graph for Interface Co-synthesis

In practice, the VHDL entity enclosing a process declares *generics* objects for those binary numbers to be instantiated later when the co-simulation program is constructed. This added benefit naturally facilitates the component reuse. Another benefit from this co-synthesis approach is that the system process graph is structurally maintained during

interface co-synthesis, which contrasts the interface synthesis in other codesign approaches. Our interface co-synthesis method obviously promotes the component reuse and alleviates the difficulty of system maintenance.

6.2.7 System Performance Evaluation (stage 6)

The work in this section corresponds to the stage 6 of our proposed codesign framework as shown in Figure 2.6. The system performance evaluation phase is actually the process involving both assembling and simulating in VHDL. The time delays acquired from performance evaluations for hardware/software components are added to the VHDL processes that are embedded in VHDL entities as shown in Appendix I. The annotated VHDL processes representing hardware/software components in the RDC system are assembled together with bus communication components and other supporting system components such as clock generators. In the process, the VHDL entities are bounded with behaviour bodies and the generics are instantiated by real parameters.

In addition to the VHDL packages and libraries introduced in previous chapters, a new VHDL package, namely *RDS_UTILITS*, has been developed. It contains definitions of special constants used in the RDC system (such as the *offset word*) and procedure/function definitions (such as encoding/decoding and error correction processes in RDS). Supported by this package, the VHDL co-simulation program is made concise, easy to develop, and less error-prone. Two typical simulation programs have been tacked to Appendix J for reference. The first program is constructed on the platform with one bus layer and the second one is with two bus layers. A series of simulations have been carried out by using the Model*Sim* EE/PLUS (version 5.2) [MTI98].

With the exploitable target architecture, the system performance is now a function of hardware cost and the bus layer. The simulations are therefore intended to discover the following facts:

- The impact on system performance as hardware cost increases (i.e. software components are gradually replaced by hardware implementations)

- The performative difference on two target architectures (one bus layer and two bus layers) as hardware cost increases

6.2.8 Analysis of Simulation Results

The simulations have been conclusive in discovery of the facts aforementioned. Before the results can be compared, we need to quantify the hardware cost. Table 6 shows hardware costs quantified by the formulas listed in the last column, which enables simulation results to be compared and charted.

Table 6.

Com. Name	Operators required	Cost	Quantification
PM_16_A/B	5 "XOR", 1 "+", 1 "COMP.", 2 "&"	22	"XOR" <==> 1
PCW_10_A/B	3 "XOR", 1 "+", 1 "COMP.", 1 "&"	19	"&" <==> 1 "COMP." <==> 5
Corrector A/B	5 "XOR", 5 "COMP.", 1 "&"	31	"+" <==> 10 "*" <==> 15
$F_{A/B}$	2 "*", 2 "+", 1 "/"	70	"/" <==> 20

Some of significant simulation results have been displayed in Table 7 through Table 10. Especially, Table 7 and 8 compose a pair of comparable sets and so do Table 9 and Table 10. In addition to tabular forms, Figure 6.9 illustrates a graphical relation between hardware cost and system performance. Its data are extracted from Table 7.

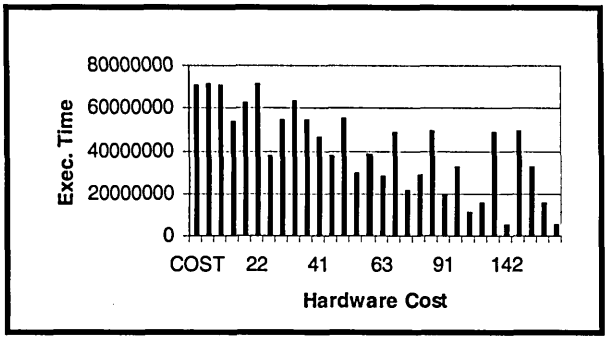


Figure 6.9 Performance vs. Hardware Cost

Figure 6.9 indicates that although it is not a linear relation between system performance and hardware cost the performance on the whole improves as the hardware cost increases (i.e. increasing number of component is implemented in hardware). Furthermore, the most effective hardware dispatch scheme is for PM_16_A/B and PCW_10_A/B to be assigned to hardware implementations, which can also be proven by the execution time at rows numbered 1, 2, 3, 7, and 8 in Table 7. This conclusion reflects a previous proposal from the hardware/software partitioning and component allocation phase.

While Figure 6.9 shows the system performance as a function of hardware cost in relation to a given target architecture, the comparison between Table 7 and Table 8 reveals how the system performance responds to variable system target architectures.

The only difference between Table 7 and 8 is the bus layer. In contrast to Table 7 that is single bus-based, Table 8 is built on two bus layers. The shadowed components in Table 8 are duly allocated to bus A whereas others are left on bus B. Besides, the hardware/software partitioning is deliberately kept in same order in both tables so as to make the rows with same number posted in different table comparable. Other system parameters remain same in both tables. They are as follows:

- The clock frequency in hardware is 20 MHZ.
- The clock frequency in ARM processor is 10 MHZ.
- All bus communication components are assumed to execute for 500 nanoseconds.

In comparison with Table 7, Table 8 shows a degree of performance improvement with two bus-layers as the hardware/software partitioning and the component allocation are carefully planned. This can be validated by a comparison with the first ten rows between Table 7 and Table 8. The improvement is, however, not as significant as we would expect. On the contrary, it becomes worse when the row number is in excess of twelve. This declining performance has resulted from the fact that the system performance depends upon both the overall performances of individual components and the communication condition over bus layers. This feature can be expressed in the formula, $P = F(c_1, c_2)$ where c_1 is the overall performances of individual components and c_2 is the communication performance. Both increases will contribute to the whole system performance P. As increasing number of components is dispatched to software implementation (i.e. c_1 decreases), the number of component connected to the bus layers is, at the same time, decreasing, which instead eases the pressure on the communication overhead (i.e. c_2 increases). The system performance could thus improve even if more components are dispatched to software implementation, which is due to the gains of communication performance outstrips the loss of overall performances contributed by individual components.

Table 7.

No.	PM_16_A	PCW_10_A	Corrector_A	F_A	PM_16_B	PCW_10_B	Corrector_B	F_B	COST	EXEC. TIME (ns)
1	H	H	H	S	H	H	H	H	214	4,982,150
2	H	H	H	S	H	H	H	S	144	4,982,550
3	H	H	H	S	H	H	S	S	113	11,016,350
4	H	H	H	S	H	S	S	S	94	19,425,950
5	H	H	H	S	S	H	S	S	91	28,427,350
6	H	H	H	S	S	S	S	S	72	28,081,650
7	H	H	S	S	H	H	H	H	183	15,506,450
8	H	H	S	S	H	H	H	S	113	15,505,250
9	H	H	S	S	H	H	S	S	82	21,444,550
10	H	H	S	S	H	S	S	S	63	29,510,450
11	H	H	S	S	S	H	S	S	60	37,690,150
12	H	H	S	S	S	S	S	S	41	37,358,250
13	H	S	S	S	H	H	H	H	164	31,980,850
14	H	S	S	S	H	H	H	S	94	31,981,150
15	H	S	S	S	H	H	S	S	63	37,919,350
16	H	S	S	S	H	S	S	S	44	45,986,350
17	H	S	S	S	S	H	S	S	41	54,166,650
18	H	S	S	S	S	S	S	S	22	53,834,750
19	S	H	S	S	H	H	H	H	161	49,403,550
20	S	H	S	S	H	H	H	S	91	49,403,350
21	S	H	S	S	H	H	S	S	60	55,101,550
22	S	H	S	S	H	S	S	S	41	63,168,550
23	S	H	S	S	S	H	S	S	38	71,349,350
24	S	H	S	S	S	S	S	S	19	71,016,950
25	S	S	S	S	H	H	H	H	142	48,712,650
26	S	S	S	S	H	H	H	S	72	48,712,650
27	S	S	S	S	H	H	S	S	41	54,423,450
28	S	S	S	S	H	S	S	S	22	62,490,450
29	S	S	S	S	S	H	S	S	19	70,670,750
30	S	S	S	S	S	S	S	S	0	70,338,850

To support the theory above, further simulations have been carried out with only one system parameter altered. The time delay in bus communication component, Synchro_Same is now extended to 2000 nanoseconds which is compared with original 500 nanoseconds. Other system parameters remain unchanged. This particular extension intends to examine the impact on system performance after the ratio of component's performance to communication's changes. The simulation results have been listed in Table 9 and Table 10 for comparison. Same as in Table 7 and Table 8, Table 9 is based on one bus layer and Table 10 on two bus layers.

Compared with Table 9, the whole performance in Table 10 significantly improves except last five rows. In addition to the improvement on overall performance, a greater range of reduction in execution time has been observed. For example, when two bus-layers are introduced, the execution time reduces 5800 nanoseconds in the first row from Table 7 to Table 8 whereas it is boosted to 246800 nanoseconds from Table 9 and Table 10.

Table 8.

No.	PM_16_A	PCW_10_A	Corrector_A	F_A	PM_16_B	PCW_10_B	Corrector_B	F_B	COST	EXEC. TIME (ns)
1	H	H	H	S	H	H	H	H	214	4,976,350
2	H	H	H	S	H	H	H	S	144	4,976,250
3	H	H	H	S	H	H	S	S	113	11,011,250
4	H	H	H	S	H	S	S	S	94	19,421,350
5	H	H	H	S	S	H	S	S	91	28,423,250
6	H	H	H	S	S	S	S	S	72	28,077,550
7	H	H	S	S	H	H	H	H	183	15,504,250
8	H	H	S	S	H	H	H	S	113	15,504,150
9	H	H	S	S	H	H	S	S	82	21,442,850
10	H	H	S	S	H	S	S	S	63	29,509,850
11	H	H	S	S	S	H	S	S	60	37,690,150
12	H	H	S	S	S	S	S	S	41	37,358,250
13	H	S	S	S	H	H	H	H	164	31,981,350
14	H	S	S	S	H	H	H	S	94	31,981,250
15	H	S	S	S	H	H	S	S	63	37,919,950
16	H	S	S	S	H	S	S	S	44	45,986,950
17	H	S	S	S	S	H	S	S	41	54,167,250
18	H	S	S	S	S	S	S	S	22	53,835,350
19	S	H	S	S	H	H	H	H	161	49,403,850
20	S	H	S	S	H	H	H	S	91	49,403,850
21	S	H	S	S	H	H	S	S	60	55,102,150
22	S	H	S	S	H	S	S	S	41	63,169,150
23	S	H	S	S	S	H	S	S	38	71,349,450
24	S	H	S	S	S	S	S	S	19	71,017,550
25	S	S	S	S	H	H	H	H	142	48,713,150
26	S	S	S	S	H	H	H	S	72	48,713,150
27	S	S	S	S	H	H	S	S	41	54,424,550
28	S	S	S	S	H	S	S	S	22	62,491,550
29	S	S	S	S	S	H	S	S	19	70,671,850
30	S	S	S	S	S	S	S	S	0	70,339,950

Table 9.

No.	PM_16_A	PCW_10_A	Corrector_A	F_A	PM_16_B	PCW_10_B	Corrector_B	F_B	COST	EXEC. TIME (ns)
1	H	H	H	S	H	H	H	H	214	5,824,350
2	H	H	H	S	H	H	H	S	144	5,617,050
3	H	H	H	S	H	H	S	S	113	11,522,450
4	H	H	H	S	H	S	S	S	94	19,830,250
5	H	H	H	S	S	H	S	S	91	28,829,350
6	H	H	H	S	S	S	S	S	72	28,483,350
7	H	H	S	S	H	H	H	H	183	16,079,750
8	H	H	S	S	H	H	H	S	113	15,871,150
9	H	H	S	S	H	H	S	S	82	21,730,950
10	H	H	S	S	H	S	S	S	63	29,721,850
11	H	H	S	S	S	H	S	S	60	37,898,550
12	H	H	S	S	S	S	S	S	41	37,566,650
13	H	S	S	S	H	H	H	H	164	32,332,550
14	H	S	S	S	H	H	H	S	94	32,130,950
15	H	S	S	S	H	H	S	S	63	37,996,350
16	H	S	S	S	H	S	S	S	44	45,990,850
17	H	S	S	S	S	H	S	S	41	54,169,650
18	H	S	S	S	S	S	S	S	22	53,837,750
19	S	H	S	S	H	H	H	H	161	49,514,950
20	S	H	S	S	H	H	H	S	91	49,409,250
21	S	H	S	S	H	H	S	S	60	55,175,050
22	S	H	S	S	H	S	S	S	41	63,171,550
23	S	H	S	S	S	H	S	S	38	71,352,350
24	S	H	S	S	S	S	S	S	19	71,018,450
25	S	S	S	S	H	H	H	H	142	48,832,750
26	S	S	S	S	H	H	H	S	72	48,712,650
27	S	S	S	S	H	H	S	S	41	54,495,450

28	S	S	S	S	H	S	S	S	22	62,491,950
29	S	S	S	S	S	H	S	S	19	70,670,750
30	S	S	S	S	S	S	S	S	0	70,338,850

Table 10.

No.	PM_16_A	PCW_10_A	Corrector_A	F_A	PM_16_B	PCW_10_B	Corrector_B	F_B	COST	EXEC. TIME (ns)
1	H	H	H	S	H	H	H	H	214	5,577,550
2	H	H	H	S	H	H	H	S	144	5,372,250
3	H	H	H	S	H	H	S	S	113	11,315,750
4	H	H	H	S	H	S	S	S	94	19,650,850
5	H	H	H	S	S	H	S	S	91	28,652,750
6	H	H	H	S	S	S	S	S	72	28,307,050
7	H	H	S	S	H	H	H	H	183	15,997,450
8	H	H	S	S	H	H	H	S	113	15,792,150
9	H	H	S	S	H	H	S	S	82	21,662,250
10	H	H	S	S	H	S	S	S	63	29,658,750
11	H	H	S	S	S	H	S	S	60	37,837,550
12	H	H	S	S	S	S	S	S	41	37,505,650
13	H	S	S	S	H	H	H	H	164	32,328,650
14	H	S	S	S	H	H	H	S	94	32,123,350
15	H	S	S	S	H	H	S	S	63	37,993,450
16	H	S	S	S	H	S	S	S	44	45,989,950
17	H	S	S	S	S	H	S	S	41	54,168,750
18	H	S	S	S	S	S	S	S	22	53,836,850
19	S	H	S	S	H	H	H	H	161	49,509,350
20	S	H	S	S	H	H	H	S	91	49,403,850
21	S	H	S	S	H	H	S	S	60	55,174,150
22	S	H	S	S	H	S	S	S	41	63,170,650
23	S	H	S	S	S	H	S	S	38	71,349,450
24	S	H	S	S	S	S	S	S	19	71,017,550
25	S	S	S	S	H	H	H	H	142	48,831,750
26	S	S	S	S	H	H	H	S	72	48,713,150
27	S	S	S	S	H	H	S	S	41	54,496,550
28	S	S	S	S	H	S	S	S	22	62,493,050
29	S	S	S	S	S	H	S	S	19	70,671,850
30	S	S	S	S	S	S	S	S	0	70,339,950

The simulation results in Table 7 through Table 10 do provide sufficient evidence for the examination of system performance. In exploitation of target architecture, we can conclude that when the communication overhead in a codesign system overtakes computational expenses in individual components extra bus layers should be considered so as to improve the system performance.

6.3 Evaluation of the Codesign Case Study

This case study provides a basis for evaluation of our proposed codesign approach. The process can be defined in the following way. During the course of codesign, the RDC system is specified in the enhanced PARSE Process Graph and Co-BSL program without prejudice on software/hardware implementations. The VHDL simulations for system functional verification and profiling have proven the design valid and provided the information heuristically for system partitioning. A number of partitioning schemes

has been planned to examine the impact on system performance as the hardware implementation and the target architecture are both exploited. The problem with evaluation of system performance is divided and conquered. The simulation in ARM SDT and the List Scheduling Algorithm have determined the performances for individual software/hardware components whereas the communication expense is decided during the annotated VHDL co-simulation. The virtual prototyping technique facilitates the smooth transition from the high level specification down to the low-level implementation. It also simplified the interface synthesis. In utilization of this technique, the system components are well preserved and reusable. Finally, the simulation results have been analyzed and the definitive conclusions have been reached.

It has exemplified the strength and practicability of the codesign methodology proposed in this project. It also demonstrates its sound theoretical bases and the potentials to be applied to the developments of other real-time embedded systems. A summary of the most significant features in our methodology is shown as follows:

- The co-specification of RDC System in Process Graph and Co-BSL program is prejudiced on neither hardware nor software implementations.
- The VHDL simulation supported by token passing protocol has verified the design in its functionality and produced abundant information heuristically for system partitioning.
- The ARM's SDT and List Scheduling Algorithm have examined performances for software and hardware components.
- The co-simulations with system target architectures based on the layered bus protocol have supplied plenty information to justify the hardware/software partitioning and component allocation.
- The VHDL packages and libraries developed in this project have been fully utilized and their practice and strength have been fully tested.

On the other hand, potential improvements emerged from this case study can be summed up as below. First of all, further to promote the co-simulation accurate to control cycles, the following problems have been envisaged:

- Current ARM's SDT tools are not accurate enough.
- The performances of bus components are not precisely evaluated.

More accurate ARM's SDT tools (nanosecond level) are required to solve the first problem, while a commercial hardware synthesis tool could be the solution to the accurate performance evaluations for bus components.

Chapter 7

Conclusions and Future Research Topics

Our research project is concluded in this chapter. A review on the research investigation is first presented, followed by a summary of the project. In the summary, contributions and dissatisfactions resulted from this project are given. Finally future research topics are suggested.

7.1 Research Investigation

The significant progress in this project is summarized below. A thorough investigation into the well-established codesign methodologies has been conducted, which covers a wide range of contents in relation to codesign approaches and their supportive tools/methods. During the course of investigation, special attentions have been focused on:

- Codesign system specification and modelling
- Hardware/software partitioning method
- System performance estimation/evaluation technique
- System target architecture

The investigation has identified shortcomings and dissatisfactions in those previously published codesign methodologies. An object-based codesign methodology: *Co-PARSE* is thus proposed. It is embodied by successive phases, guidelines, and techniques to support reaching a solution particularly to a real-time embedded system and at the same time respecting the criteria of system performance and hardware cost. Tools have been developed to support the use of the methodology.

To demonstrate the strength and practicability of the proposed codesign methodology an extensive case study has been carried out. During the course of codesign of a new RDC System, the proposed codesign phases are applied and the guidelines and tools that are designed in support of the methodology are fully utilized. Especially, the VHDL simulation with token passing protocol and the co-simulation with the virtual prototyping have proven practical in verification of system functionality and evaluation

of performance satisfaction. Besides, the extensive VHDL simulations have produced large volume of simulation results to support the hardware/software partitioning and prove the system target architecture able to be exploited.

7.2 Summary of the Project

This project addressed the shortcomings and problems in other codesign methodologies published to date. It then proposed a new objected-based codesign methodology, which is indeed a combination of methods and techniques from diverse fields. It has authentically engineered structured and coherent sets of methods, guidelines and tools for solving the problems in codesign of hardware/software. Besides, the methodology has itself been evaluated by an extensive case study. The results from the case study have proven conclusive.

Contributions from this research project are hereby claimed as follows:

1. Analytical Contributions
    - The system-level **C**odesign **B**ehaviour **S**pecification **L**anguage (Co-BSL) has been designed. It is used to capture the overall dynamic aspects of codesign system and the performative constraints (see Chapter 3). The most significant feature with Co-BSL is its implementation neutrality. The Co-BSL not only specifies the hierarchical structure of codesign system and its communication paths but also provides a high level language for the codesigner to describe the sequential behaviour of each primitive (i.e. non-decomposable) process object.
    - The concept of exploiting system target architecture to improve the system performance has been proposed in this thesis (see Chapter 5). Furthermore, designing the asynchronous bus protocol and the layered bus communication structure have materialized this concept. None of such approach has been reported before in this research area.
    - The co-synthesis method particularly used for synthesis of hardware/software interface has been devised. The method guides through smooth transition from path definitions in Co-BSL description to the system implementations, based on the layered bus structure (see Chapter 5 and 6)
2. Developmental Contributions

- Six VHDL packages and one VHDL library have been developed and tested. They provide a portable platform for future codesign research projects and they are necessary to support the evaluation of the proposed codesign approach. They include:
  * Four VHDL packages designed to support, in hardware/software partitioning phase, functional verification and system profiling, which facilitate the template conversion from Co-BSL communication channels to the VHDL simulation program (see Chapter 3)
  * One VHDL package and one VHDL library, which provide communication components and support the communication protocol in the interface co-synthesis phase (see Chapter 5)
  * One VHDL package, which implements the major algorithms in the case study, the *Radio Data Computing System* (see Chapter 6)
- The *ARM SDT Tool Kit* and the *List Scheduling Algorithm* have been integrated into our co-design methodology at the system performance evaluation phase (see Chapter 5)

3. Evaluative Contributions

The proposed *Co-PARSE* codesign methodology has itself been evaluated in a case study. It has been applied to a real-time embedded system and used to design the *Radio Data Computing System* (see Chapter 6). The case study has demonstrated the strength and suitability of the proposed codesign methodology. The simulation results have proven:

- The VHDL simulations with token passing protocol can both verify the system design and produce the heuristics for the hardware/software partitioning and the component allocation scheme.
- The annotated VHDL co-simulations can serve the purpose of performance evaluation for codesign system. It can also help examine the impact on system performance when both hardware cost and system target architecture are exploited.

4. Disseminating Contributions

Part of the subject matters addressed in this thesis has been published in the following conference proceedings and technical reports: [CLJ98a][CLJ98b][CLJ97] [CLJ96a][CLJ96b][LJC95][CLJ95a][CLJ95b][CRL00] (see References).

Despite these substantial contributions to knowledge, further improvements could be made and some questions remain unanswered. Whereas those unanswered questions are dealt with in next section, the following discussions focus on potential areas for improvement.

First of all, the data structure *RECORD* in the Co-BSL programming language looks rather awkward. It could have been better replaced by a CLASS as in an object-oriented programming language. The problem stems from the inherited imperfection in PARSE due to its lack of inheritance and polymorphism. This is the very reason we labeled our methodology an object-based methodology instead of the object-oriented methodology.

Second, in the co-specification phase of our codesign approach we provide user with five communication paths: synchronous, asynchronous, bi-directional, broadcast, and wire. Although conversions of these communication paths to VHDL and C programs introduced in chapter 3 have established a smooth transition route to components interface synthesis and functional verification plus system profiling, the broadcast communication path in the co-specification is currently not supported in the *Bus Interface Module*, i.e. the low level components interface synthesis does not support the broadcast communication path. Consequently the user has to design their own VHDL components to implement this communication path supported in the co-specification.

Third, though the ARM's debugger can theoretically examine the execution time accurate to individual assembly instruction, the debugger we used in this project is a freeware version because of which the precision of the simulations has been compromised. The problem with the debugger results in the fact that the examination of the system performance has to be based on the whole function body instead of the individual instruction, which is ideally discussed in Chapter 5.

Finally the performances of the standard bus communication components such as (A)Synchronous Channel Gateways we designed in Chapter 5 have not yet precisely evaluated in a hardware synthesis algorithm or a hardware synthesis tool. The execution times of these components are currently assumed as a given time period. While this

treatment does not affect selections of those hardware/software partitioning schemes that can effectively improve the system performance, the accuracy of the co-simulation results should be taken as caution.

7.3 Potential Research Directions

While considerable progress has been achieved in this project, some questions remain unanswered. We present them below as potential directions for further research.

First, as pointed in this thesis, there has been very limited research conducted in codesign society in relation to exploration of system target architecture to improve system performance. Although this research has laid a foundation towards its final solution, the conceptual model of the target architecture with a layered bus communication structure proposed in this project only represents one of the possibly practical models, which could be adopted as the target architecture in a codesign discipline. In general, it is worthwhile to investigate further the possibility of reclassifying system target architectures suitable for the codesign applications. The classification could be conducted according to relevant application domains characterised by their computational complexity and respective constraints. The system target architecture could thus be delineated in a scientific and systematic (objective) manner rather than through experience or prejudice (subjective).

Second, the system-partitioning criteria in this research are solely focused on the hardware cost and system performance. Other criteria are, however, also potentially exploitable in the system-partitioning phase in this codesign approach. For example, in addition to the execution time, the software code/data size can be obtained additionally from the ARM SDT debugging process. This value-added property from the integration of ARM SDT tools could benefit dispatching components to software/hardware implementation with the minimum memory occupation in the processor. The criteria of system evaluation can then be extended to hardware cost and(/or) system performance and(/or) software memory occupation i.e. system code and data size. On the other hand, as indicated in the performance evaluation for hardware component, when the List Scheduling Algorithm is applied, a number of alternative hardware resource constraints were left unused in the system performance evaluation phase. Only the resource

constraint derived from the ASAP scheduling has been used, which leaves extra scope for hardware resources to be further exploited in trading for the system performance.

Third, the operations: hardware/software partitioning, selection of system target architecture, and the allocation of communication channels and hardware/software components can all strike an impact on system performance. Close relations among them can have clearly been perceived. Current research is however merely concentrated on one particular aspect that is uniformly the hardware/software partitioning. A future research direction aimed at revelation of those relations could introduce a new dimension in exploration of design space and bring about great benefit to codesign products. This new dimension changes the codesign framework as well. An indicative codesign framework following this new direction has been illustrated in Figure 7.1, which forms the basis for further work. This unorthodox framework emphasizes the mutual impacts regarding hardware/software partitioning, communication channel and component allocation, and selection of system target architecture. The bi-directional arrows in the graph imply the interactive activities between them.

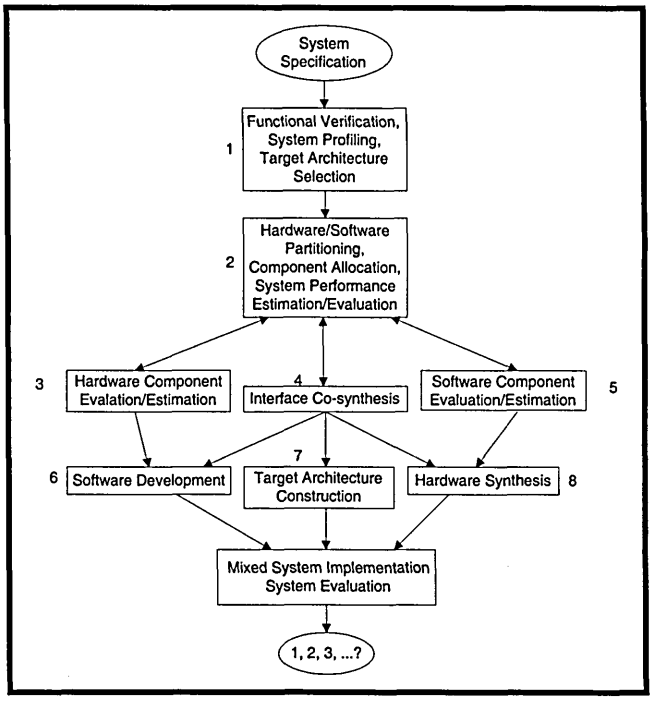


Figure 7.1 New Codesign Framework

Finally, as hinted in Chapter 3 and five, our proposed codesign approach has the potential for automation. Particularly, the Co-BSL specification could be automatically transformed into VHDL and C programs that can in turn be compiled by VHDL and C

compilers to create the executable codes. The conversion from Co-BSL to VHDL/C is composed by two separate issues. The one is the constructs. The other is the communication channel. Although in this research the conversion from Co-BSL description to VHDL and C programs are manually done following guidelines set up in Chapter 3, the conversions are in fact established on mapping from Co-BSL design constructs and individual statements to the equivalent VHDL/C counterparts. On the other hand, the conversion of communication channels is also built on mapping from Co-BSL communication channels to the construction of bus communication components plus numbering, which has been described in Chapter 5. The conversion is therefore a compilation. As seen in Chapter 3, the syntax of Co-BSL has been written and examined by the YACC compiler. The remaining task for this purpose is to produce the formal descriptions of semantics for the Co-BSL language, which describes how the source language (Co-BSL) can be compiled into its target language (VHDL/C). A Co-BSL compiler would be straightforward to develop.

Another automation issue is to integrate a commercial hardware synthesizer into our codesign approach. The synthesizer can automatically accomplish the hardware implementation. In addition, the synthesizer can schedule the VHDL description for hardware component and provide vital information in relation to hardware cost and system performance without manual operation in List Scheduling Algorithm as shown in Chapter 5.

References

[AF+97] A. Allara, S. Filipponi, W. Fornaciari. F. Salice and D. Sciuto, Improving Design Turnaround Time via Two-Levels HW/SW Co-Simulation, *Proceedings of International Conference on Computer Design: VLSI in Computers and Processors*, Austin, Texas, USA, 1997, pp. 400-405

[AG97] Samir Agrawal and Rajesh K. Gupta, "Data-flow Assisted Behavioural Partitioning for Embedded Systems", *Proceedings of 34th Design Automation Conference*, Anaheim, CA, USA, 1997

[Ala90] Alan M. Davis, Software Requirements analysis & specification, Prentice-Hall International, Inc. 1990

[And95] Andrew S. Tanenbaum, Distributed Operating systems, Prentice-Hall International, 1995

[ARM97] Advanced RISC Machines Ltd. (ARM), ARM Software Development Toolkit, User Guide & Reference Guide, 1997

[Ash96] Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann, 1996

[Ayl92] H. Aylor et. al., "The Integration of Performance and Functional Modelling in VHDL", Performance and Fault Modelling with VHDL, J. Schoen, ed., Prentice Hall, Englewood Cliffs, N. J., 1992, pp.22-145

[Bak93] Louis Baker, VHDL Programming with Advanced Topics, John Wiley & Sons, Inc. 1993

[BCW91] Timothy C. Bell, John G. Cleary and Ian H. Witten, Text compression, Prentice Hall, Englewood Cliffs, N. J., 1991

[BE97] Matthias Bauer and Wolfgang Ecker, "H/S Co-Simulation in a VHDL-based Test Bench Approach", *Proceedings of 34th Design Automation Conference*, Anaheim, CA, USA, 1997

[Ben90] J. P. Bennett, Introduction to Compiling Techniques: a First Course using ANSI C, LEX and YACC, McGraw-Hill, 1990

[BFS94] Antoniazzi, A. Balboni, W. Fornaciari and D. Sciuto, "The Role of VHDL within the TOSCA H/S Codesign Framework", *Proceedings of EURO-VHDL'94* , 1994, pp. 612-617

[BFS96] Antoniazzi, A. Balboni, W. Fornaciari and D. Sciuto, "Co-synthesis and Co-simulation of Control-Dominated Embedded Systems", Design Automation for Embedded Systems, Kluwer Academic Publisher, 1996, pp. 257-289

[BG97] Smita Bakshi and Daniel D. Gajski, "Hardware/Software Partitioning and Pipelining", *Proceedings of 34th Design Automation Conference*, Anaheim, CA, USA, 1997

[Bha98] Murali Bharathala, "VHDL Times", viewlogic Systems, Inc. , 1998

[BHL+94] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, special issue on "Simulation Software Development", April 1994, Vol. 4, pp. 155-182

[BKK+99] J-Y. Brunel, E.A. de Kock, W.M. Kruijtzer, H.J.H.N. Kenter, W.J.M. Smiths, "Communication Refinement in Video Systems On Chip", *Proceedings of Seventh International Workshop on Hardware/Software Codesign*, Rome, Italy, 1999 , pp. 142 146

[Bra94] Warrick Bradney, "PARSE Code Translator", Master Degree Project Report, University of Wollongong, November 1994

[Bri92] British standard of CENELEC EN 50067: Specification for Radio Data System (RDS), 1992

[BSV93] Dr. K Buchenrieder, A. Sedllmeier, C. Veith, "HW/SW Co-Design With PRAMs Using CODES, *11th IFIP International Conference*, 1993

[Buc94] Klaus Buchenrieder, Hardware/Software Co-design *An Annotated Bibliography*, IT Press Hartenstein, 1994

[Cal93] Jean Paul Calvez, Embedded Real-time Systems, Wiley, 1993

[Cam90] R. Camposano, "From Behaviour to Structure: High-Level Synthesis", *IEEE Design & Test of Computers*, Oct. 1990, pp. 8-19.

[CDK01] George Coulouris, Jean Dollimore, and Tim Kindberg, Distributed Systems, Concepts and Design, Addison-Wesley, 2001

[CGJ+94] Massimiliano Chiodo, Paolo Giusto, Attila Jurecska, Harry C. Hsieh, Alberto Sangiovanni-Vincentelli, and luciano Lavagno, "Hardware-Software Codesign of Embedded Systems", *IEEE Micro*, Vol. 14, No. 4, August 1994, pp. 26-36.

[CLJ95a] Jianming CAI, D W Lloyd, and I E Jelly, "A Survey of Low-power Multimedia Systems from the Prospective of Hardware/Software Codesign", *Sheffield Hallam University Technical Report Series CRC-95-3*

[CLJ95b] Jianming CAI, D W Lloyd, and I E Jelly, "A Brief Survey of the Recent Developments in Hardware-Software Codesign", *Sheffield Hallam University Technical Report Series CRC-95-7*

[CLJ96a] Jianming CAI, D W Lloyd, and I E Jelly, "Conversion from PARSE Notations to VHDL Program Templates", *Sheffield Hallam University Technical Report Series CRC-96-3*

[CLJ96b] Jianming CAI, D W Lloyd, and I E Jelly, "An Investigation into High Level Specification for Codesign Systems", *Sheffield Hallam University Technical Report Series CRC-96-2*

[CLJ97] Jianming CAI, D W Lloyd, and I E Jelly, "Modelling Mobile Communication Systems in the Integrated PARSE and VHDL Environment", *Proceedings of The 9th International Conference on Wireless Communications*, 9-11 July 1997, Calgary, Alberta, Canada

[CLJ98a] Jianming CAI, D W Lloyd, and I E Jelly, "An Evaluation of Partitioning Techniques for Hardware/Software Codesign", *Proceedings of The Fourth Chinese Automation Conference in the UK (CACUK'98)*, 19-20 September, 1998, Leicester, UK, (ISBN 0953389006, Published by Pacilantic International 1998)

[CLJ98b] Jianming CAI, D W Lloyd, and I E Jelly, "Modelling and Simulating Radio Data Systems", *Proceedings of The 1st International Symposium on Communication Systems and Digital Signal Processing*, 6-8 April 1998, Sheffield, UK, (ISBN 0-86339-7719, Published by Sheffield Hallam University Press and Learning Centre 1998)

[CRL00] Jianming CAI, I. E. Ritchie, and David W. Lloyd, "A Simulation Technique for Codesign of Hardware/Software", *Proceedings of 2000 Summer Computer Simulation Conference*, July 16-20, 2000, Vancouver, British Columbia, Canada, (ISBN 1-56555-208-3, Published by The Society for Computer Simulation International, Nov. 2000)

[CW93] Paul Camposano and Wayne Wolf (Eds.), High-Level VLSI Synthesis, Kluwer Academic Publishers, 1991

[CY91] Peter Coad and Edward Yourdon, Object-Oriented Analysis, Prentice-Hall International, Inc. 1991, ISBN 0-13-630013-8

[Dav+94] David A. Patterson et. al. Computer Organization & Design the hardware/ software interface, Morgan Kaufmann, 1994

[Edw93] M. Edwards, "A Development system for Hardware/Software Co-Synthesis using FPGAs", *Second IFIP International Workshop on HW-SW CoDesign*, May, 1993

[Edw97] M. Edwards, "Software Acceleration Using Coprocessor: is it worth the effort ?", *Proceedings of the Fifth Intl. Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1997, pp. 135-139

[EF94] M. Edwards and John Forrest, "A Development Environment for the Cosynthesis of Embedded Software/Hardware Systems", *Proceedings of European Design Automation Conference*, 1994, pp. 469-473

[EF96] M. Edwards and John Forrest, "A Practical Hardware Architecture to Support Software Acceleration, *Microprocessors and Microsystems*, Vol. 20, No.3, pp. 167-174, 1996

[EFW97] M. Edwards, John Forrest, and A. E. Whelan, "Acceleration of Software Algorithms Using Hardware/Software Co-Design Techniques, *Journal of Systems Architecture*, Vol. 42, No. 9/10, pp. 697-707, 1997

[EH92] R. Ernst, J. Henkel, "Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction", *IEEE Workshop on Hardware-Software Co-Design*, Estes Park, Colorado, Oct. 1992

[EHB93] R. Ernst, J. Henkel and Thomas Benner, "Hardware-Software Cosynthesis for Microcontrollers", *IEEE Design & Test of Computers*, December, 1993

[Emb01] The Embedded Marketplace, http://www.microcontroller.com, Jan. 2001

[EPD94] Petru Eles, Zebo Peng, and Alexa Doboli, "VHDL system-Level Specification and Partitioning in a Hardware/Software Co-Synthesis Environment", *Proceedings of the Third International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1994, pp. 49-55

[FFSS97] Allara, S. Filipponi, W. Fornaciari, F. Salice and D. Sciuto, "A Flexible Model for Evaluating the Behaviour of H/S Systems", *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1997, pp. 109-114

[For95] John Forrest, "Implementation-Independent Descriptions Using an Object-Oriented Approach", Personal Communications, 1995

[FPGA00] FPGA Related WWW Links, {http://www.mrc.uidaho.edu/fpga/fpga.html}, July 2000

[FS96] Balboni, W. Fornaciari and D. Sciuto, Hardware/Software Co-Design, Kluwer Academic Publisher, 1996, pp. 265-294

[FS99] W. Fornaciari and D. Sciuto, "HW/SW Co-design of Embedded Systems", *Proceedings of European Design Automation Conference*, 1999, pp. 344-355

[Fur96] S.B. Furber, ARM System Architecture, Addison-Wesley, 1996

[Gaj92] Daniel D. Gajski et al. High-Level Synthesis, Introduction to Chips and System Design, Kluwer Academic Publishers, 1992

[Gaj94] Daniel D. Gajski et al. Specification and Design of Embedded Systems, PTR Prentice Hall, 1994

[GB94] J. P. Gray and W. Bradney, "Behavioural Specification Language for Parse process Graphs", Tech. Report Parse-TR4a, Computer Science Dept., University of Wollongong, Australia, 1994

[GG94] Gert Goossens et al., Design of heterogeneous ICs for mobile and personal communication systems. *Proceedings of ACM/IEEE International Conference on Computer-Aided Design*, 1994, pp. 524-3

[GGJ95] I Gorton, J P Gray and I E Jelly, "Object Based Modelling of Parallel Programs", *IEEE Parallel and Distributed Technology Journal*, Vol. 3, No. 2, 1995, IEEE Computer Society Press (1995)

[Gia90] Joseph Di Giacomo, Digital bus handbook, McGraw-Hill, 1990

[GJC94] I Gorton, I E Jelly and T Soon Chan, "Engineering High Quality Parallel Software Using PARSE", *Proceedings of CONPAR*, Sept 1994, Linz, Austria, Springer Verlag LNCS (1994)

[GJG93] Gorton, I, Jelly, I E and Gray, J P, "Parallel software engineering with PARSE", in Pro. COMPSAC-17, IEEE International Computer Software

and Applications Conference, November 1993, Phoenix, Arizona, USA (1993), PP 123-130

[GM93] Rajesh K. Gupta and Giovanni De Micheli, "Hardware-Software Cosynthesis for Digital Systems", *IEEE Design & Test of Computers*, Vol.10, No.3, September 1993, pp. 29-41

[GJM94] Rajesh K. Gupta, Claudionor N. Coelho Jr., and Giovanni De Micheli, "Program Implementation Schemes for Hardware-Software systems", *IEEE Computer*, Vol. 27, No. 1, January 1994, pp. 48-55

[Gra94] I. Graham, Object Orientated Methods Second Edition, Addison-Wesley, 1994

[Gray95] Jonathan Gray, "Relational and Textual Representations of PARSE Process Graphs", Technical Report PARSE-TR-3b, June 1995

[GVN94] Daniel D. Gajski, Frank Vahid and Sanjiv Narayan, A System-Design Methodology: Executable-Specification Refinement, *Proceeding of European Design Automation Conference*, 1994, pp. 458-463

[GVNG94] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan and Jie Gong, Specification and Design of Embedded Systems, PTR Prentice Hall, 1994

[GZGH00] A. Gerstlauer, S. Zhao, D. Gajski and A. Horak, "SpecC System-Level Design Methodology Applied to the Design of a GSM Vocoder", *Proceedings of Ninth Workshop on Synthesis and System Integration of Mixed Technologies*, Kyoto, Japan, April 2000

[HB97] Ken Hines and Gaetano Borriello, "Dynamic Communication Models in Embedded Systems Co-Simulation", *Proceedings of 34th Design Automation Conference*, Anaheim, CA, USA, 1997

[HE97] J. Henkel and R. Ernst, "A Hardware/Software Partitioner Using a Dynamically Determined Granularity", *Proceedings of 34th Design Automation Conference*, IEEE Computer Society Press, 1997

[HE98] J. Henkel and R. Ernst, "High-Level Estimation Techniques for Usage in Hardware/Software Co-Design", *Proceedings of Asia and South Pacific Design Automation Conference,* IEEE Computer Society Press, 1999

[HG97] L. Hu, I. Gorton, "A Performance Prototyping Approach to Designing Concurrent Software Architectures", in Software Engineering for Parallel and Distributed Systems, Boston, 17-18 May 1997

[Hoa85] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, N. J., 1985

[Hua85] J. P. Huang, "Modelling of software partition for distributed real-time applications", *IEEE Transactions on Software Engineering*, Vol. 11, no. 10, pp. 1113-1126, Oct. 1985

[HW96] Junwei Hou and W. Wolf, "Process Partitioning for Distributed Embedded Systems", *Proceedings of the Fourth Intl. Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1996, pp. 70-76

[ICSP93] Proceedings of *the First International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1993

[IEEE92] IEEE Design & Test of Computers, September 1992, pp.3-30

[IEEE94] Institute of Electrical and Electronics Engineers, IEEE Standard VHDL Language Reference Manual, IEEE, 1994

[IEEE98] Institute of Electrical and Electronics Engineers, IEEE Standard VHDL Language Reference Manual, IEEE, 1998

[JDV92] Roy, N. Jumar, R. Dutta and R. Vemuri, " DSS: A Distributed High-Level Synthesis System", *IEEE Design and Test of Computers*, June 1992

[JE+94] Axel Jantsch, Peeter Ellervee, Johnny Öberg, Ahmed Hemani and Hannu Tenhunen, "Hardware/software Partitioning and Minimizing Memory Interface Traffic", *Proceedings of European Design Automation Conference*, 1994, pp. 226-231

[JJ93] K. P. Juliussen and E. Juliussen, *The 6th Annual Computer Industry Almanac*, The Reference Press, Austin, TX, 1993

[Joh80] Stephen C. Johnson, Yet Another Compiler's Compiler, User Guide by Bell Laboratories, 1980

[Kai93] Kai Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, Inc. 1993

[KAJW93] Sanjaya Kumar, James H. Aylor, Barry W. Johnson, and Wm. A. Wulf, "A Framework for Hardware/Software Codesign", *IEEE Computer*, Vol. 26, No. 12, December 1993, pp.39-45

[KAJW96] Sanjaya Kumar, James H. Aylor, Barry W. Johnson, and Wm. A. Wulf, "The Codesign of Embedded Systems: A Unified Hardware/Software Representation", Kluwer Academic Publishers, 1996

[Kal96] Asawaree Prabhakar Kalavade, *System-Level Codesign of Mixed Hardware-Software Systems*, Ph.D. thesis, University of California, Berkeley, 1996

[KL93] Asawaree Kalavade and Edward A. Lee, "A Hardware-Software Codesign Methodology for DSP Application", *IEEE Design & Test of Computers*, Vol. 10, No. 3, September 1993, pp. 16-28

[KL96] Chang, A. Kalavade and E. A. Lee, Hardware/Software Co-Design, Kluwer Academic Publisher, 1996, pp. 187-212

[KM98] Peter Voigt Knudsen and Jan Madsen, Communication Estimation for Hardware/Software Codesign, *Proceedings of the Sixth Intl. Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1998, pp. 55-59

[KMA97] Robert H. Klenke, Moshe Meyassed, James H. Aylor, "An Integrated Design Environment for Performance and Dependability Analysis", *Proceedings of the ACM Design Automation Conference*, June 1997, pp. 184-189

[KN97] Russ Klein and Ross Nelson, Seamless CVE™ Hardware/Software Co-Verification Technology, *http://www.mentorg.com/codesign*, 1997

[Kum94] Gupta, Rajesh Kumar, *Cosynthesis of Hardware and Software for Digital Embedded Systems*, Ph.D. thesis, Stanford University, 1994

[Kum96] Gupta, Rajesh Kumar, "A Framework for Interactive Analysis of Timing Constraints", *Proceedings of the Fourth International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1996

[LG96] Liu, I. Gorton, "Modelling Dynamic Distributed System Structures In PARSE", in 4th EUROMICRO Workshop on Parallel and Distributed Processing, IEEE Computer Society Press, Braga, Portugal, January 24-26th 1996, pp. 352-359

[LJC95] D W Lloyd, I E Jelly and Jianming CAI, "Evaluation of PARSE for High-Level Codesign Specification", *proceedings of International Conference on Recent Advances in Mechatronics*, 14-16 August 1995, Istanbul, Turkey

[LLS99] Marcello Lajolo, Mihai Lazarescu and Alberto Sangiovanni-Vincentelli, "A Compilation-based Software Estimation Scheme for Hardware/ Software Co-Simulation", *Proceedings of Seventh International Workshop on Hardware/ Software Codesign*, Rome, Italy, 1999 , pp. 85-89

[LLV98] Jie Liu, Marcello Lajolo, and Alberto Sangiovanni-Vinventelli, "Software Timing Analysis Using HW/SW Cosimulation and Instruction Set Simulator", *Proceedings of the Sixth Intl. Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1998, pp. 65-69

[LMB92] John R. Levine, Tony Mason and Doug Brown, LEX and YACC, O'Reilly & Associates, 1992

[LO96] Mike Loukides & Andy Oram, Programming with GNU Software, O'Reilly & Associates, 1996

[LSU89] Roger Lipsett, Carl Schaefer, and Cary Ussery, VHDL: Hardware Description and Design, Kluwer Academic Publishers, 1989

[LW97] Li and W. Wolf, "Allocation of Multirate Systems on Multiprocessors with Memory Hierarchy Modelling and Optimization", *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1997

[Mic94] Giovanni De Micheli, "Hardware-Software Codesign", *IEEE Micro*, August 1994, pp. 10-16

[Mic99] Giovanni De Micheli, "Hardware Synthesis from C/C++ Models, *Proceedings of the Design Automation and Test in Europe Conference*, 1999, pp. 382-383

[MP92] Mouly and Marie-B. Pautet, The GSM System, Anne-Laure and Guillaume, 1992

[MTI98] Model Technology Incorporated, Model*Sim* EE/PLUS Reference Manual, 1998

[MW90] Peter L Mothersole and Norman W White, Broadcast Data System, Teletext and RDS, Butterworths & Co. (Publishers) Ltd., 1990

[NG97] Mark Nelson and Jean-Loup Gailly, The Data Compression Book, M&T Books, 1997

[Nie91] Matthias Niemeyer, "Simulation of Heterogeneous Models with a Simulator Coupling System", *Proceedings of SCS 1991 European Simulation Multi. Conference*, June 1991, pp. 388-393

[PB96] P.J. Plauger and Jim Brodie, Standard C: a reference, Prentice-Hall, 1996

[Per94] Douglas L. Perry, VHDL , McGraw-Hill, Inc. 1994

[PF92] Martin K. Purvis and David W. Franke, "An Overview of Hardware/Software Codesign", *Proceedings of IEEE Symposium on VLSI and Systems*, 1992

[PLCV97] Claudio Passerone, Luuciano Lavagno, Massimiliano Chiodo and Alberto Sangiovanni-Vincentelli, "Fast Hardware/Software Co-simulation for Virtual Prototyping and Trade-off Analysis", *Proceedings 34th DAC*, Anaheim, CA, USA, 1997

[PPT00] J. M. Paul, S. N. Peffers, and D. E. Thomas, "Frequency Interleaving as a Co-Design Scheduling Paradigm", *Proceedings of International Workshop on Hardware/Software Co-Design*, May 2000

[Pre94] Roger Pressman, Software Engineering: a practitioner's approach, McGraw-Hill, Inc. 1994

[PTWP99] J. M. Paul, D. E. Thomas, S. J. Weber, and S. N. Peffers, "Hardware and Software as Dual Languages for Computer System Modelling", IEEE Computer Society Annual Workshop on VLSI, April 1999

[PW72] W. Wesley Peterson and E.J. Weldon, Error-Correcting Codes, MIT Press, Cambridge Mass. Second Edition, 1972

[Rao90] Ramesh Rao, A Building Block Approach to Performance Modelling Using VHDL, Centre for Semicustom Integrated Systems, University of Virginia, 1990

[RDS98] RDS Forum - News, WWW address: http://www.org.uk/rdsupgrade. html, 14th, Jan. 1998

[Red95] An Introduction to GSM, Artech House, Inc. 1995

[RPJL+96] S. Russo, S.J. Pateman, I.E. Jelly, D.W. Lloyd, P.C. Collingwood, C. Savy, "PARSE and DISC Integration for Parallel Software Development", *IEEE International Conference on Algorithms and Architectures for Parallel Processing*, Singapore, June 1996, IEEE-CS Press

[RSJ95] Russo S., Savy C. & Jelly I., "Petri Net Modelling of PARSE Designs", EUROMICRO '95 Como, Italy, Sept. 1995

[Sad95] D R Sadler, " Hardware/Software Codesign of Parallel Processing Arrays", MSc Dissertation, Sheffield Hallam University, Sheffield UK, Sept. 1995

[SB91] Mani B. Srivastava and Robert W. Brodersen, "Rapid-Prototyping of Hardware and Software in a Unified Framework", *Proceedings of International Conference on Computer Aided Design* , IEEE Press, 1991, pp. 152-155

[SC94] Sony Corporation. DVW-700P/700WSP, Digital BETACAM One-piece Camcorder. *Sony Corporation*, 1994

[Sch92] Joel M. Schoen, Editor, Performance and Fault Modelling with VHDL, Prentice-Hall, Englewood Cliffs, N. J., 1992

[SJA93] Maximo H. Salinas, Barry W. Johnson, and James H. Aylor, "Implementation-Independent Model of an Instruction Set Architecture in VHDL", in IEEE Design & Test of Computers, Vol. 10, No. 3, September 1993, pp. 42-54.

[Som93] V. Someren, ARM Risc Chip A Programmer's Guide, Addison-Wesley, 1993

[Spe84] Specification of the Radio Data System, RDS, for VHF/FM Sound Broadcasting, Specification Tech. 3244-E, European Broadcasting Union, Technical Centre, Geneva, 1984

[Sri93] Mani Bhuahan Srivastava, *Rapid-Prototyping of Hardware and Software in a Unified Framework*, Ph.D. thesis, University of California, Berkeley, 1993

[TA92] Takashi Asaida et al. Digital Signal Processing for Broadcast TV Cameras, *International Broadcasting Convention*, 1992

[TAS93] Donald E. Thomas, Jay K. Adams, and Herman Schmit, "A Model and Methodology for Hardware-Software Codesign", *IEEE Design & Test of Computers*, Vol. 10, No. 3, September 1993, pp. 6-15

[Tex88] Texas Instruments, Bus Interface Circuits: Applications and Data Book, Texas Instruments Ltd., 1988

[Tsa00] Jeff Tsay, "A Code Generation Framework for Ptolemy II", ERL Technical Report UCB/ERL No. M00/25, Dept. EECS, University of California, Berkeley, CA 94720, May, 2000

[VME82] VMEbus Manufacturers Group, VMEbus Specification Manual, Aug. 1982

[VG92] Frank Vahid and Daniel d. Gajski, " Specification Partitioning for System Design", in *29th ACM/IEEE Design Automation Conference*, 1992

[VG95] Frank Vahid and Daniel d. Gajski, "Clustering for improved system-level functional partitioning", *Proceedings of IEEE International Symposium on System Synthesis*, pp. 28-33, 1995

[WC91] Robrt A. Walker and Paul Camposano, "A Survey of High-Level Synthesis Systems", Kluwer Academic Publishers, 1991

[WDW94] Nam S. WOO, Alfred E. Dunlop, and Wayne Wolf, "Codesign from Cospecification ", *IEEE Computer*, Vol. 27, No. 1, January 1994, pp. 42-55

[Wol94] Wayne H. Wolf, "*Hardware-Software Co-Design of Embedded Systems*", *Proceedings of the IEEE*, Vol. 82, No. 7, July 1994, pp. 967-987

[YEBH93] W. Ye, R. Ernst, Th. Benner, and J. Henkel, "Fast Timing Analysis for Hardware-Software Co-Synthesis", *Proceedings of International Conference on Computer Design*, 1993

[YW95] Ti-Yen Yen and Wayne Wolf, "Performance Estimation of Distributed Embedded Systems", *Proceedings of ICCD'95*, IEEE Computer Society Press, 1995

Appendix A

```
%{
/*      This is a YACC compatible syntax definition for the Codesign-BSL      */
/*      (Co-BSL). Descriptions in Co-BSL can be readily converted into        */
/*      VHDL or C  program.                                                   */
/*                                                                            */
/*       Compared with the ordinary BSL, the Co-BSL is featured               */
/*       as follows:                                                          */
/*      1. all Co-BSL elements are user accessible, i.e. it does not          */
/*        separate the textual portion from the user accessible ones          */
/*        as in BSL [Bra94].                                                  */
/*      2. its program portions are convertible to VHDL and C program         */
/*        counterparts.                                                       */
/*      3. a new communication channel by name WIR is introduced to model     */
/*        the communication between hardware components.                      */
/*      4. during cosynthesis stage, communication channels are               */
/*        transformed into a configuration with distributed target            */
/*        architectures for performance evaluations.                          */
/*      5. its data type and expression are flourished to comprise            */
/*        those specially for hardware properties.                            */
/*      6. statements asserting system constraints particularly on            */
/*        hardware aspects are enhanced.                                      */
/*      7. conventional delimiters are used to make the program               */
/*        more readable.                                                      */
/*      8. both procedure and function are enclosed.                          */
/*      9. object-based features are preserved but those irrelevant to        */
/*        codesign process are dropped.                                       */
/*      10. other minor changes are introduced.                               */

%}
%token  CODESIGN END_CODESIGN CONSTANT
%token  PATH SYNC ASYN BROD BIDI WIRE
%token  PRIMITIVE PROCESS END_PROCESS
%token  INPORT OUTPORT CONSTRUCTOR
%token  DET NON_DET CONC VARIABLE
%token  INT REAL BYTE BOOLEAN BIT
%token  OCTAL HEX CHAR TIME
%token  BEGIN END WAIT ON FOR UNTIL
%token  BREAK CONTINUE RETURN SKIP STOP
```

```
%token  IF THEN ELSE_IF END_IF
%token  CASE IS END_CASE ON MOD REM
%token  TRUE FALSE REM UMINUS
%token  AND OR NAND NOR XOR XNOR NOT ABS
%token  SLL SRL SLA SRA ROL ROR
%token  WHILE DO END_WHILE FOR LOOP END_FOR
%token  CLASSES D_CLASS END_CLASS
%token  CONNECT_PATHS FROM TO CONNECT_PORTS
%token  CALL EXTERNAL INPORTS OUTPORTS
%token  INTERFACE END_INTERFACE
%token  ARRAY RECORD END_RECORD AFTER
%token  NAME NUMBER EXECUTION
%token  FUNCTION PROCEDURE IO_NAME OF
%token  FUNCTION_SERVER DATA_SERVER
%token  CONTROL_PROCESS
%token   RECEIVE  SYN-SEND  ASYN-SEND BR-SEND

%start codesign

%left  '+' '-'
%left  '*' '/'
%right UMINUS

%%
codesign:
                CODESIGN codesign_name
                    constants
                    paths
                    primitives
                    classes
                    externals
                    executions
                    connections
                END_CODESIGN
;
codesign_name:
                undef_name
;
constants:
                /* empty */
```

```
                | CONSTANT constant_list ';'
;
constant_list:
                  constant_dec
                | constant_list ';' constant_dec
;
constant_dec:
                  undef_name '=' NUMBER
;
paths:
                  /* empty */
                | PATH path_list ';'
;
path_list:
                  path_def
                | path_list ';' path_def
;
path_def:
                  undef_name path_type type
;
path_type:
                  SYNC | ASYN | BROD | BIDI | WIRE
;
primitives:
                  /* empty */
                | PRIMITIVE primitive_list ';'
;
primitive_list:
                  primitive_def
                | primitive_list ';' primitive_def
;
primitive_def:
                  PROCESS process_name OF class_type time_indication
                      inports
                      outports
                      constructors
                      variables
                      function_declaration
                      procedure_declaration
                      main_sequence
```

```
                END_PROCESS
;
process_name:
                undef_name
;
time_indication:
                /* empty */
               | '{' expression '}'
;
inports:
                /* empty */
               | INPORT port_def_list ';'
;
port_def_list:
                port_descriptor
               | port_def_list ';' port_descriptor
;
port_descriptor:
                port_id path_type type
;
port_id:
                undef_name
;
outports:
                /* empty */
               | OUTPORT port_def_list ';'
;
constructors:
                /* empty */
               | CONSTRUCTOR constructor_list ';'
;
constructor_list:
                constructor_def
               | constructor_list ';' constructor_def
;
constructor_def:
                NAME constructor_type '(' priority_list ')'
;
constructor_type:
                DET | NON_DET | CONC
```

```
;
priority_list:
                priority_entry
               | priority_list ';' priority_entry
;
priority_entry:
                NAME
;
variables:
                /* empty */
               | VARIABLE variable_list ';'
;
variable_list:
                variable_def
               | variable_list ';' variable_def
;
variable_def:
                undef_name_list ':' type
;
type:
                prim_type | composite_type
;
prim_type:
                INT | REAL | BYTE | BOOLEAN
               | OCTAL | HEX  | CHAR | TIME | BIT
;
composite_type:
                array_declarator | record_declarator
;
array_declarator:
                ARRAY subscript prim_type
;
subscript:
                '[' NUMBER ']' | '[' NUMBER ']' subscript
;
record_declarator:
                 RECORD element_declaration_list
                 END_RECORD
;
element_declaration_list:
```

```
                element_declaration
              | element_declaration_list element_declaration
;
element_declaration:
                undef_name ':' type ';'
;
undef_name_list:
                undef_name
              | undef_name_list ',' undef_name
;
undef_name:
                NAME    /* previously defined name */
;
def_name:
                NAME    /* undefined name */
;
function_declaration:
                /* empty */
              | FUNCTION function_specifier '(' association_list ')'
                RETURN type
                main_sequence ';'
;
procedure_declaration:
                /* empty */
              | PROCEDURE procedure_specifier '(' association_list ')'
                main_sequence ';'
;
function_specifier:
                undef_name
;
procedure_specifier:
                undef_name
;
association_list:
                undef_name
              | association_list ',' undef_name
;
main_sequence:
                BEGIN statements END
;
```

```
statements:
                    statement
                  | statements ';' statement
;
statement:
                    /* empty */
                  | wait_statement | BREAK | CONTINUE
                  | RETURN | SKIP | STOP | assignment
                  | signal_assignment | condition_statement
                  | case_statement | loop | proc_call
                  | io_operation
;
wait_statement:
                    WAIT wait_tail
;
wait_tail:
                    ON name_list
                  | UNTIL expression
                  | FOR expression
;
name_list:
                    def_name
                  | name_list ',' def_name
;
assignment:
                    element ':' '=' expression
;
element:
                    def_name | def_name subscript_expr
;
signal_assignment:
                    def_name '<' '=' '=' expression wave_form
;
wave_form:
                    /* empty */
                  | AFTER expression
;
actural_parameter:
                    identifier_list
;
```

```
identifier_list:
                identifier
              | identifier ',' identifier_list
;
identifier:
                NAME | NUMBER
;
subscript_expr:
                '[' expression ']'
              | '[' expression ']' subscript_expr
;
expression:
                relation
              | expression AND  relation
              | expression OR   relation
              | expression NAND relation
              | expression NOR  relation
              | expression XOR  relation
              | expression XNOR relation
;
relation:
                shift_expression
              | relation  '<'  shift_expression
              | relation '=' '<' shift_expression
              | relation  '>'  shift_expression
              | relation '=' '>' shift_expression
              | relation '<' '>' shift_expression
              | relation  '='  shift_expression
;
shift_expression:       simple_expression
                | SLL simple_expression
                | SRL simple_expression
                | SLA simple_expression
                | SRA simple_expression
                | ROL simple_expression
                | ROR simple_expression
;
simple_expression:
                term
              | sign term
```

```
                | simple_expression adding_operator term
;
sign:
                '+' %prec UMINUS
                | '-' %prec UMINUS
;
adding_operator:
                '+' | '-' | '&'
;
term:
                factor
                | term multi_operator factor
;
multi_operator:
                '*' | '/' | REM | MOD
;
factor:
                primary | primary '^' primary %prec UMINUS
                | NOT primary %prec UMINUS | ABS primary %prec UMINUS
;
primary:
                NAME | literal | '(' expression ')' | function_call
;
literal:
                NUMBER | TRUE | FALSE
;
function_call:
                function_name '(' actural_parameter ')'
;
function_name:
                def_name
;
condition_statement:
                IF expression THEN statements END_IF
                | IF expression THEN statements else_part END_IF
;
else_part:
                ELSE_IF expression THEN statements
                | else_part ELSE_IF expression THEN statements
;
```

```
case_statement:
                CASE expression IS caselist END_CASE
;
caselist:
                ON expression statements
               | caselist ON expression statements
;
loop:
                WHILE expression DO statements END_WHILE
               | FOR for_expression LOOP statements END_FOR
;
for_expression:
                '(' expression ';' expression ';' expression ')'
;
proc_call:
                CALL def_name '(' arglist ')'
;
arglist:
                /* empty */
               | single_arg | arglist ',' single_arg
;
single_arg:
                expression
;
io_operation:
               io_name '(' expression_list ')'
;
io_name:
              RECEIVE | SYN-SEND | ASYN-SEND | BR-SEND
;
classes:
                /* empty */
               | CLASSES class_defs
;
class_defs:         class_def
               | class_defs ';' class_def
;
class_def:      D_CLASS class_name OF class_type
                      inports
                      outports
```

```
                    constructors
                    paths
                    executions
                    connections
                    portconnects
                  END_CLASS
;
class_name:
                  undef_name
;
class_type:
                  FUNCTION_SERVER
                | DATA_SERVER
                | CONTROL_PROCESS
;
externals:
                  /* empty */
                | EXTERNAL  external_list ';'
;
external_list:
                  external_dec
                | external_list ';' external_dec
;
external_dec:
                  INTERFACE NAME interface_io END_INTERFACE
;
interface_io:
                  /* empty */
                | INPORTS port_descriptor
                | OUTPORTS port_descriptor
;
executions:
                  /* empty */
                | EXECUTION execution_list ';'
;
execution_list:
                  execution_dec
                | execution_list ';' execution_dec
;
execution_dec:
```

```
                NAME ':' def_class_name class_params
;
def_class_name:
                def_name
;
class_params:
                /* empty */
               | '(' expression_list ')'
;
expression_list:
                expression
               | expression_list ',' expression
;
connections:
                /* empty */
               | CONNECT_PATHS path_instances ';'
;
path_instances:
                path_instance
               | path_instances ';' path_instance
;
path_instance:
                path_name ':' FROM instance TO instance
;
path_name:
                def_name
;
instance:
                vname
;
vname:
                def_name
               | def_name vector
;
vector:
                '[' expression ']'
;
portconnects:
                /* empty */
               | CONNECT_PORTS  port_connect_list ';'
```

```
;
port_connect_list:
                    port_connect
                  | port_connect_list ';' port_connect
;
port_connect:
                    formal_param TO vname '.' vname
                  | vname '.' NAME TO formal_param
;
formal_param:
                    def_name
;

%%
```

Appendix B

1. Bus Resolution Function (BRF)

```
FUNCTION protocol(input: token_vector) RETURN token is
 variable source_token: token := def_source_token;
 variable sink_token: token := def_sink_token;
 variable i: integer;
 variable no_source, no_sink: boolean := true;
begin

for i in input'low to input'high loop
 if (input(i).status = active_source) then
  source_token.status := input(i).status;
  source_token.color := input(i).color;
  no_source := false;
 elsif (input(i).status = active_sink) then
  sink_token := input(i);
  no_sink := false;
 elsif (input(i).status = inactive_source) then
  source_token := input(i);
  no_source := false;
 elsif (input(i).status = inactive_sink) then
  sink_token := input(i);
  no_sink := false;
 end if;
end loop;

if no_source then
  return sink_token;
elsif no_sink then
  return source_token;
elsif(source_token.status = active_source) then
  if (sink_token.status = active_sink) then
     return sink_token;
  else
     return source_token;
  end if;
elsif (sink_token.status = active_sink) then
     return source_token;
else
     return sink_token;
end if;

end protocol;
```

2. Synchronous Sending Procedure

```
PROCEDURE syn_transmit(signal T: inout token; variable tt: in token;
                delay:time; wit: time) IS
variable temp, temp1:  token;
begin
if tt.color.condi = TRUE then
  temp  := tt;
  if not(token_removed(T)) then
    wait until(token_removed(T) or time_out(wit));
  end if;
  if token_removed(T) then
```

```
   place_token(T, temp, delay);
   wait until (token_acked(T)) or time_out(wit);
  end if;
  if token_acked(T) then
   release_token(T);
  end if;
 end if;
end syn_transmit;
```

3. Synchronous Receiving Procedure

```
PROCEDURE syn_receive(signal T: inout token; variable tt: out token;
                delay:time; wit: time) IS
variable temp: token;
begin
if not(token_present(T)) then
  wait until(token_present(T) or time_out(wit));
end if;
if token_present(T) then
  tt := T;
  ack_token(T, delay);
  wait until (token_released(T)) or time_out(wit);
end if;
if token_released(T) then
  remove_token(T);
end if;
end syn_receive;
```

4. Asynchronous Buffer Procedure

```
PROCEDURE asyn_buffer(signal T_in: inout token; signal T_out: inout token;
                que_point: inout queptr_tk; delay1: time; delay2: time;
                wit: time) IS
variable temp: token;
variable tmp: token;
variable tmp1: integer;
begin
if token_present(T_in) then
  tmp := T_in;
  ack_token(T_in, delay2);
  in_que_tk(que_point, tmp);
  wait until (token_released(T_in) or time_out(wit));
  if token_released(T_in) then
    remove_token(T_in);
  end if;
end if;
if token_removed(T_out) then
  number_of_que_tk(que_point, tmp1);
  if ( tmp1 > 0 ) then
    out_que_tk(que_point,tmp);
    place_token(T_out, tmp, delay1);
  end if;
end if;
if token_acked(T_out) then
   release_token(T_out);
end if;
end asyn_buffer;
```

5. Synchronous Procedure for Broadcast Communication

```
PROCEDURE bro_syn(signal T: inout token; wit: time) IS
begin
if not(token_present(T)) then
  wait until token_present(T) or time_out(wit);
end if;
if token_present(T) then
  ack_token(T);
  wait until (token_released(T) or time_out(wit));
end if;
if token_released(T) then
  remove_token(T);
end if;
end bro_syn;
```

6. Broadcast Buffer Procedure

```
PROCEDURE bro_buffer(signal T_in: inout token; signal T_out: inout token;
                que_point: inout queptr_tk; wit: time) IS
variable tmp: token;
variable tmpp: token;
variable tmp1: integer;
begin
if token_present(T_in) then
  tmp := T_in;
  ack_token(T_in);
  in_que_tk(que_point, tmp);
  wait until (token_released(T_in) or time_out(wit));
  if token_released(T_in) then
    remove_token(T_in);
  end if;
end if;
if token_removed(T_out) then
     number_of_que_tk(que_point,tmp1);
     if (tmp1 > 0 ) then
       out_que_tk(que_point,tmpp);
       place_token(T_out, tmpp);
     end if;
end if;
if token_acked(T_out) then
  release_token(T_out);
end if;
end bro_buffer;
```

Appendix C

1. The Co-BSL Design for Handover

```
-----------------------------------------------------------------------------------------------------------------
-- The following Co-BSL design skeletonizes the Handover process in GSM communication
-- To focus on the usage of Co-BSL, sequential behavioural descriptions inside primitives and
-- some of program details are not included because they are not essential in this example.
-- The Co-BSL code has been broken into several files that can be invoked with
-- "#includefilename" and inserted into the desired places.
-----------------------------------------------------------------------------------------------------------------

CODESIGN Handover

-- The Handover Process in GSM Mobile Communications

PATH
Notification   ASYN  RECORD name: ARRAY [32] CHAR;
                            number: INT;
                            area: BYTE;
                     END_RECORD;
Maintenance  BIDI    RECORD  control: ARRAY [32] BIT;
                             number: INT;
                             area: BYTE;
                      END_RECORD;
Control_1      BIDI   ARRAY [114]  BIT;
Control_2      BIDI   ARRAY [114]  BIT;
Traffic_1      BIDI   ARRAY [148]  BIT;
Monitor_1      WIRE   ARRAY [88]   BIT;
Traffic_2      BIDI   ARRAY [148]  BIT;
Monitor_2      WIRE   ARRAY [88]   BIT;

PRIMITIVE

#include "co.prims"

CLASSES

#include "co.clases"

EXTERNAL

#include "co.externs"

EXCUTION
b:   BSC;
b1:  BTS1;
b2:  BST2;
m:   MS;
g:   GMSC;
o:   OMC;

CONNECT_PATHS
Notification: FROM b.outp1 TO g.inp1;
Maintenance: FROM o.outp1 TO b.inp1;
```

```
Control_1: FROM b.outp2 TO b1.inp1;
Control_2: FROM b.outp3 TO b2.inp1;
Traffic_1: FROM b1.outp1 TO m.inp3;
Monitor_1: FROM instance TO m.inp1;
Traffic_2: FROM b2.outp1 TO m.inp4;
Monitor_2: FROM instance TO m.inp2;

END_CODESIGN

-- "co.prims"
-- primitive process descriptions

PROCESS BSC OF CONTROL_PROCESS
INPORTS
inp1 BIDI RECORD  control: ARRAY [32] BIT;
                  number: INT;
                  area: BYTE;
          END_RECORD;

OUTPORTS
outp1 ASYN  RECORD name: ARRAY [32] CHAR;
                   number: INT;
                   area: BYTE;
            END_RECORD;
outp2 BIDI   ARRAY [114] BIT;
outp3 BIDI   ARRAY [114] BIT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

PROCESS BTS1 OF FUNCTION_SERVER
INPORTS
inp1 BIDI ARRAY [114] BIT;

OUTPORTS
outp1 BIDI ARRAY [148] BIT;
   outp2 WIRE ARRAY [88] BIT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

PROCESS BTS2 OF FUNCTION_SERVER
```

```
INPORTS
inp1 BIDI ARRAY [114] BIT;

OUTPORTS
outp1 BIDI ARRAY [148] BIT;
    outp2 WIRE ARRAY [88] BIT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

PROCESS Transceiver OF FUNCTION_SERVER
INPORTS
inp1 BIDI ARRAY [148] BIT;
inp2 BIDI ARRAY [148] BIT;
inp3 WIRE  ARRAY [88] BIT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

PROCESS Monitor OF FUNCTION_SERVER
INPORTS
inp1 WIRE  ARRAY [88] BIT;
inp2 WIRE  ARRAY [88] BIT;

OUTPORTS
outp1 WIRE  ARRAY [88] BIT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

-- "co.clases"
-- decomposable class
```

```
D_CLASS Mobile_Station OF FUNCTION_SERVER
INPORTS
inp1 WIRE  ARRAY [88] BIT;
inp2 WIRE  ARRAY [88] BIT;
inp3 BIDI   ARRAY [148] BIT;
inp4 BIDI   ARRAY [148] BIT;

CONSTRUCTORS
     -- constructor declaration

PATH
Monitor_data WIRE ARRAY [88] BIT;

EXCUTION
t:   Transceiver;
mr:  Monitor;

CONNECT_PATHS
Monitor_data: FROM mr.outp1 TO t.inp3;
inp1 TO mr.inp1;
inp2 TO mr.inp2;
inp3 TO t.inp1;
inp4 TO t.inp2;

CONNECT_PORTS

END_CLASS

-- "co.externs"
-- external interface objects

INTERFACE GMSC
INPORTS
inp1 ASYN  RECORD name: ARRAY [32] CHAR;
                  number: INT;
                  area: BYTE;
           END_RECORD;

END_INTERFACE

INTERFACE OMC
OUTPORTS
outp1 BIDI   RECORD  control: ARRAY [32] BIT;
                     number: INT;
                     area: BYTE;
             END_RECORD;

END_INTERFACE
```

2. The VHDL Program Converted from its Co-BSL Design

```
-----------------------------------------------------------------------------------------------------------------
-- Following the guidelines outlined in Chapter 3, the Co-BSL program shown earlier
-- in this appendix has been converted into the VHDL program shown below. Special
-- attention has been given to the preservation of object-based features within Co-BSL
-- design. All primitives in the original Co-BSL program have been converted
-- into VHDL entities and their architecture bodies. They are supposed to be stored
-- in the  component library WORK for future reuse.
-----------------------------------------------------------------------------------------------------------------

entity BSC is
   port (
        in_bidi1: in RECORD  control: ARRAY [32] BIT;
                             number: INT;
                             area: BYTE;
                   END_RECORD;
        in_bidi2: out RECORD  control: ARRAY [32] BIT;
                              number: INT;
                              area: BYTE;
                    END_RECORD;
        out1_asyn: out RECORD name: ARRAY [32] CHAR;
                               number: INT;
                               area: BYTE;
                     END_RECORD;
        out2_bidi1: in   ARRAY [114] BIT;
        out2_bidi2; out ARRAY [114] BIT;
        out3_bidi1: in   ARRAY [114] BIT;
        out3_bidi2: out ARRAY [114] BIT );
end BSC;

architecture Behavior of BSC is
begin
    process
        variable
            -- variable declaration
        begin
            -- behavioral description
       end process
end Behavior;

entity BTS1 is
   port (
         in_bidi1: in      ARRAY [114] BIT;
         in_bidi2: out     ARRAY [114] BIT;
         out1_bidi1: in   ARRAY [148] BIT;
         out1_bidi2: out ARRAY [148] BIT;
         out2_wire; out  ARRAY [88] BIT );
end BTS1;

architecture Behavior of BTS1 is
begin
    process
        variable
            -- variable declaration
        begin
```

```
        -- behavioral description
      end process
end Behavior;

entity BTS2 is
  port (
       in_bidi1: in      ARRAY [114] BIT;
       in_bidi2: out     ARRAY [114] BIT;
       out1_bidi1: in    ARRAY [148] BIT;
       out1_bidi2: out ARRAY [148] BIT;
       out2_wire; out  ARRAY [88] BIT );
end BTS2;

architecture Behavior of BTS2 is
begin
   process
      variable
         -- variable declaration
      begin
        -- behavioral description
      end process
end Behavior;

entity Transceiver is
  port (
       in1_bidi1: in   ARRAY [148] BIT;
       in1_bidi2: out ARRAY [148] BIT;
       in2_bidi1: in   ARRAY [148] BIT;
       in2_bidi2: out ARRAY [148] BIT;
        in3_wire: in   ARRAY [88] BIT );
end Transceiver;

architecture Behavior of Transceiver is
begin
   process
      variable
         -- variable declaration
      begin
         -- behavioral description
      end process
end Behavior;

entity Monitor is
  port (
       in1_wire: in     ARRAY [88] BIT;
       in2_wire: in     ARRAY [88] BIT;
       out1_wire: out ARRAY [88] BIT );
end Monitor;

architecture Behavior of Monitor is
begin
   process
      variable
         -- variable declaration
      begin
         -- behavioral description
      end process
end Behavior;

-- Two externals (GMSC and OMC) are too converted into VHDL entity declarations
```

```
-- and architecture bodies. They are stored in the library WORK.


entity Ext_GMSC is
   port (
        in1_asyn: in RECORD  name: ARRAY [32] CHAR;
                             number: INT;
                             area: BYTE;
                    END_RECORD );
end EXT_GMSC;

architecture Behavior of EXT_GMSC is
begin
    process
        variable
            -- variable declaration
        begin

         -- the behavior depends on the design of test bed for simulation.

      end process
end Behavior;


entity EXT_OMC is
   port (
        out_bidi1: in RECORD   control: ARRAY [32] BIT;
                               number: INT;
                               area: BYTE;
                    END_RECORD;
        out_bidi2: out RECORD  control: ARRAY [32] BIT;
                               number: INT;
                               area: BYTE;
                     END_RECORD );
end EXT_OMC;

architecture Behavior of EXT_OMC is
begin
   process
        variable
            -- variable declaration
        begin

           -- the behavior depends on the design of test bed for simulation.

      end process
end Behavior;


-- The class Mobile_Station is converted into a VHDL entity declaration and its
-- architecture body. The conversion reuses the components, which have been
-- converted from the primitives and stored in the library WORK.


entity CLASS_Mobile_Station is
   port (
        in1_wire: in   ARRAY [88] BIT;
        in2_wire: in   ARRAY [88] BIT;
        in3_bidi1: in  ARRAY [148] BIT;
```

```
        in3_bidi2: out ARRAY [148] BIT;
        in4_bidi1: in  ARRAY [148] BIT;
        in4_bidi2: out ARRAY [148] BIT );
end CLASS_Mobile_Station;

architecture Behavior of CLASS_Mobile_Station is

component Transceiver
   port (
        in1_bidi1: in  ARRAY [148] BIT;
        in1_bidi2: out ARRAY [148] BIT;
        in2_bidi1: in  ARRAY [148] BIT;
        in2_bidi2: out ARRAY [148] BIT;
        in3_wire: in   ARRAY [88] BIT );
end Transceiver;

component Monitor
   port (
        in1_wire: in    ARRAY [88] BIT;
        in2_wire: in    ARRAY [88] BIT;
        out1_wire: out ARRAY [88] BIT );
end Monitor;

signal Monitor_data_wire: ARRAY [88] BIT;

begin
   C1: Transceiver port map (in3_bidi1, in3_bidi2, in4_bidi1, in4_bidi2, Monitor_data_wire );
   C2: Monitor  port map (in1_wire, in2_wire, Monitor_data_wire );
end Behavior;



-- Now comes the Co-BSL program "Handover" (top-level description), converted

-- into a VHDL entity declaration and architecture body. Its components have

-- already been stored in the library WORK.


entity Handover is
-- no port declaration !
end Handover;

architecture Behavior of Handover is

component BSC
   port (
        in_bidi1: in RECORD  control: ARRAY [32] BIT;
                             number: INT;
                             area: BYTE;
                   END_RECORD;
        in_bidi2: out RECORD  control: ARRAY [32] BIT;
                              number: INT;
                              area: BYTE;
                    END_RECORD;
        out1_asyn: out RECORD  name: ARRAY [32] CHAR;
                               number: INT;
                               area: BYTE;
                     END_RECORD;
        out2_bidi1: in  ARRAY [114] BIT;
        out2_bidi2; out ARRAY [114] BIT;
        out3_bidi1: in  ARRAY [114] BIT;
```

```
        out3_bidi2: out ARRAY [114] BIT );
end BSC;

component BTS1
   port (
        in_bidi1: in      ARRAY [114] BIT;
        in_bidi2: out     ARRAY [114] BIT;
        out1_bidi1: in    ARRAY [148] BIT;
        out1_bidi2: out ARRAY [148] BIT;
        out2_wire; out  ARRAY [88] BIT );
end BTS1;

component BTS2
   port (
        in_bidi1: in      ARRAY [114] BIT;
        in_bidi2: out     ARRAY [114] BIT;
        out1_bidi1: in    ARRAY [148] BIT;
        out1_bidi2: out ARRAY [148] BIT;
        out2_wire; out  ARRAY [88] BIT );
end BTS2;

component CLASS_Mobile_Station
   port (
        in1_wire: in    ARRAY [88] BIT;
        in2_wire: in    ARRAY [88] BIT;
        in3_bidi1: in   ARRAY [148] BIT;
        in3_bidi2: out ARRAY [148] BIT;
        in4_bidi1: in   ARRAY [148] BIT;
        in4_bidi2: out ARRAY [148] BIT );
end CLASS_Mobile_Station;

component Ext_GMSC
   port (
        in1_asyn: in RECORD  name: ARRAY [32] CHAR;
                             number: INT;
                             area: BYTE;
                    END_RECORD );
end EXT_GMSC;

component EXT_OMC
   port (
        out_bidi1: in RECORD  control: ARRAY [32] BIT;
                              number: INT;
                              area: BYTE;
                     END_RECORD;
        out_bidi2: out RECORD  control: ARRAY [32] BIT;
                               number: INT;
                               area: BYTE;
                      END_RECORD );
end EXT_OMC;

signal Notification RECORD name: ARRAY [32] CHAR;
                           number: INT;
                           area: BYTE;
                   END_RECORD;
signal Maintenance1 RECORD control: ARRAY [32] BIT;
                           number: INT;
                           area: BYTE;
                     END_RECORD;
signal Maintenance2 RECORD control: ARRAY [32] BIT;
```

```
                                number: INT;
                                area: BYTE;
                        END_RECORD;
signal Control_11  ARRAY [114]  BIT;
signal Control_12  ARRAY [114]  BIT;
signal Control_21  ARRAY [114]  BIT;
signal Control_22  ARRAY [114]  BIT;
signal Traffic_11   ARRAY [148]  BIT;
signal Traffic_12   ARRAY [148]  BIT;
signal Monitor_1   ARRAY [88]   BIT;
signal Traffic_21   ARRAY [148]  BIT;
signal Traffic_22   ARRAY [148]  BIT;
signal Monitor_2   ARRAY [88]   BIT;

begin

T1: component BSC
    port map (Maintenance1, Maintenance2, Notification,
              Control_11, Control_12, Control_21, Control_22);

T2: component BTS1
    port map (Control_12, Control_11, Traffic_11, Traffic_12, Monitor_1);

T3: component BTS2
    port map (Control_22, Control_21, Traffic_21, Traffic_22, Monitor_2);

T4: component CLASS_Mobile_Station
    port map (Monitor_1, Monitor_2, Traffic_12, Traffic_11, Traffic_22, Traffic_21);

T5: component Ext_GMSC
    port map (Notification);
T6: component EXT_OMC
    port map (Maintenance2, Maintenance1);

end Behavior;
```

Appendix D

1. VHDL Program

- Data Source

```
USE std.textio.ALL;
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
USE WORK.token_definition.ALL;
USE WORK.token_passing.ALL;
USE WORK.par_vhdl_conversion.ALL;
USE WORK.arith_pack.ALL;

entity data_source is
     port(sent_signals: inout token);
end data_source;

architecture behave_data_source of data_source is

signal data_temp: token_res;

file test_counts: byte_file; -- "test.counts"
file test_bits:   bit_file;  -- "test.bits"

begin

encoder: process
variable temp:    token;
variable tmp:     integer;
variable tmp_bit: bit;
variable counts_int:  unsigned_int;
variable counts_char: unsigned_char;
variable c_vector:    bit_vector(0 to 15):=(others =>'0');

begin

--/ sending off scaled_counts /--
file_open(test_counts, "test.counts", read_mode);
for i in 0 to 255 loop
   read(test_counts, counts_char);
   c_vector:= (others =>'0');
   c_vector(0 to 7):= unsignchar_to_bv(counts_char, 8);
   counts_int:= bv_to_unsignint(c_vector);
   temp.color.data1:= counts_int;
   temp.color.condi:= true;
   syn_transmit(sent_signals, temp, 2 ns, 99999 ns);
end loop;

file_close(test_counts);

--/ sending off arith_bits /--
file_open(test_bits, "test.bits", read_mode);
loop1:
loop
  c_vector:= (others =>'0');
  for i in 0 to 15 loop
     if not endfile(test_bits) then
        read(test_bits, tmp_bit);
```

```
      c_vector:= c_vector(1 to 15) & tmp_bit;
    else
      tmp:= i;
      exit loop1;
    end if;
 end loop;
 temp.color.data1:= bv_to_unsignint(c_vector);
 temp.color.condi:= true;
 syn_transmit(sent_signals, temp, 2 ns, 99999 ns);
end loop loop1;

--/ sending off remaining bits /--
for i in tmp to 15 loop
   c_vector:= c_vector(1 to 15) & '0';
end loop;
temp.color.data1:= bv_to_unsignint(c_vector);
temp.color.condi:= true;
syn_transmit(sent_signals, temp, 2 ns, 99999 ns);
wait for 2 ns;

file_close(test_bits);
wait;

end process;

end behave_data_source;
```

- Control_block

```
USE std.textio.ALL;
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
USE WORK.token_definition.ALL;
USE WORK.token_passing.ALL;
USE WORK.par_vhdl_conversion.ALL;
USE WORK.arith_pack.ALL;

entity control_block is
     port(in_16:    inout token;
          out_bit:  inout token;
          out_16:   inout token;
          out_rest: inout token);
end control_block;

architecture behave_control_block of control_block is

begin
control_block:  process
variable t_out: token;
variable tmp: unsigned_int;
variable t_times: bit:= '0';
variable t_count: integer:= 0;
variable marks1: integer:= -1;
variable marks2: integer:= -1;
variable marks3: integer:= -1;
variable t_in_counts: integer:= 0;
variable c_received: bit_vector(0 to 15);
file datafile: text open write_mode is "control.txt";
variable l: line;

begin
```

```
if (marks1 = -1) then
  --/ scaled_counts /--
  syn_receive(in_16, t_out, 99999 ns);
  t_out.color.condi:= true;
  syn_transmit(out_16, t_out, 99999 ns);
  t_in_counts:= t_in_counts + 1;
  if (t_in_counts > 255) then
     marks1:= 0;
  end if;
elsif (marks2 = -1) then
  --/ first arith_coded 16-bit message /--
  syn_receive(in_16, t_out, 99999 ns);
  tmp:= t_out.color.data1;
  c_received:= unsignint_to_bv(tmp, 16);
  for i in 0 to 15 loop
     t_out.color.data2:= c_received(i);
     t_out.color.condi:= true;
     syn_transmit(out_bit, t_out, 99999 ns);
  end loop;
  marks2:= 0;
elsif (marks3 = -1) then
  --/ the remaining arith_coded message /--
  syn_receive(in_16, t_out, 99999 ns);
  tmp:= t_out.color.data1;
  c_received:= unsignint_to_bv(tmp, 16);
  for i in 0 to 15 loop
     t_out.color.data2:= c_received(i);
     t_out.color.condi:= true;
     syn_transmit(out_rest, t_out, 99999 ns);
  end loop;
end if;

--/ counting invoking time /--
write(l, t_times, right, 1);
t_times:= not t_times;
if t_count >= 29 then
   write(l, NOW, right, 15);
   writeline(datafile,l);
   t_count:= -1;
end if;
t_count:= t_count + 1;

end process;

end behave_control_block;
```

- Model_Building

```
USE std.textio.ALL;
USE work.ESSENTIAL_DEFINITIONS.ALL;
USE work.token_definition.ALL;
USE work.token_passing.ALL;
USE work.par_vhdl_conversion.ALL;
USE work.arith_pack.ALL;

entity building_model is
     port(in_counts: inout token; total_1: inout token);
end building_model;

architecture behave_building_model of building_model is
```

```
begin

building_model: process
variable totals: array_totals;
variable t_out: token;
variable scaled_counts: array_char;
variable tmp: unsigned_int;
variable t_times: bit:= '0';
variable t_count: integer:= 0;
variable t_in_count: integer:= 0;
variable t_out_count: integer:= -1;
variable c_received: bit_vector(0 to 15);
file datafile: text open write_mode is "building.txt";
variable l: line;

begin
  if (t_out_count = -1) then
    --/ receiving scaled_counts from the decoder /--
    syn_receive(in_counts, t_out, 99999 ns);
    tmp:= t_out.color.data1;
    c_received := unsignint_to_bv(tmp, 16);
    scaled_counts(t_in_count):= bv_to_unsignchar(c_received(0 to 7));
    t_in_count:= t_in_count + 1;
    if (t_in_count > 255) then
      t_in_count:= 0;
      t_out_count:= 0;
      --/ building totals /--
      build_totals(scaled_counts, totals);
    end if;
  elsif (t_in_count = 0) then
      --/ sending totals to the expander /--
      t_out.color.data1 := totals(t_out_count);
      t_out.color.condi := TRUE;
      syn_transmit(total_1, t_out, 99999 ns);
      t_out_count:= t_out_count + 1;
      if t_out_count > 257 then
        t_out_count:= -1;
      end if;
  end if;

  --/ counting invoking time /--
  write(l, t_times, right, 1);
  t_times:= not t_times;
  if t_count >= 29 then
    write(l, NOW, right, 15);
    writeline(datafile,l);
    t_count:= -1;
  end if;
  t_count:= t_count + 1;

end process;

end behave_building_model;
```

- Expander

```
USE std.textio.ALL;
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
USE WORK.token_definition.ALL;
```

```
USE WORK.token_passing.ALL;
USE WORK.par_vhdl_conversion.ALL;
USE WORK.arith_pack.ALL;

entity expander is
      port(in_totals: inout token;
           in_code:   inout token;
           out_data:  inout token;
           in_data:   inout token;
           out_store: inout token);
end expander;

architecture behave_expander of expander is

begin

expander: process
variable s: symbol;
variable c: unsigned_int;
variable t_temp: token;
variable t_times: bit:= '0';
variable t_count: integer:= 0;
variable totals: array_totals;
variable scale_count: array_char;
variable t_in_counts: integer:= 0;
variable t_out_counts: integer:= -1;
variable high, low, code, count: unsigned_int;
file datafile: text open write_mode is "expanding.txt";
variable l: line;

begin
  if (t_out_counts = -1) then
     --/ receiving totals /--
     syn_receive(in_totals, t_temp, 99999 ns);
     totals(t_in_counts):= t_temp.color.data1;
     t_in_counts:= t_in_counts + 1;
     if (t_in_counts > 257) then
        t_in_counts:= 0;
        t_out_counts:= 0;
        --syn_receive(in_code, t_temp, 99999 ns);
        --code:= t_temp.color.data1;
        initialize_arithmetic_decoder(in_code, code);
        low:=  0;
        high:= 65535;
     end if;
  elsif (t_in_counts = 0) then
     --/ decoding process /--
     s.scale:= totals(257);
     count:= get_current_count(s, high, low, code);
     convert_symbol_to_int(count, c, s, totals);
     t_temp.color.data1:= c;
     t_temp.color.condi:= true;
     syn_transmit(out_store, t_temp, 99999 ns);

     --/ sending off the data /--
     t_temp.color.data1:= s.low_count;
     t_temp.color.condi:= true;
     syn_transmit(out_data, t_temp, 99999 ns);
     t_temp.color.data1:= s.high_count;
     t_temp.color.condi:= true;
```

```
      syn_transmit(out_data, t_temp, 99999 ns);
      t_temp.color.data1:= s.scale;
      t_temp.color.condi:= true;
      syn_transmit(out_data, t_temp, 99999 ns);
      t_temp.color.data1:= high;
      t_temp.color.condi:= true;
      syn_transmit(out_data, t_temp, 99999 ns);
      t_temp.color.data1:= low;
      t_temp.color.condi:= true;
      syn_transmit(out_data, t_temp, 99999 ns);
      t_temp.color.data1:= code;
      t_temp.color.condi:= true;
      syn_transmit(out_data, t_temp, 99999 ns);

      --/ receiving the data-back /--
      syn_receive(in_data, t_temp, 99999 ns);
      high:= t_temp.color.data1;
      syn_receive(in_data, t_temp, 99999 ns);
      low:= t_temp.color.data1;
      syn_receive(in_data, t_temp, 99999 ns);
      code:= t_temp.color.data1;
    end if;

    --/ counting invoking time /--
    write(l, t_times, right, 1);
    t_times:= not t_times;
    if t_count >= 29 then
      write(l, NOW, right, 15);
      writeline(datafile,l);
      t_count:= -1;
    end if;
    t_count:= t_count + 1;

end process expander;

end behave_expander;
```

- Remover

```
USE std.textio.ALL;
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
USE WORK.token_definition.ALL;
USE WORK.token_passing.ALL;
USE WORK.par_vhdl_conversion.ALL;
USE WORK.arith_pack.ALL;

entity remover is
      port(in_bits:      inout token;
           in_unsigned:  inout token;
           out_unsigned: inout token);
end remover;

architecture behave_remover of remover is

begin
remover: process
variable t_temp: token;
variable s: symbol;
variable range1: integer;
variable bit_tmp: bit;
```

```
variable t_times: bit:= '0';
variable t_count: integer:= 0;
variable mark1: integer:= -1;
variable high, low, code: unsigned_int;
variable bv16_high: bit_vector(0 TO 15);
variable bv16_low:  bit_vector(0 TO 15);
variable bv16_code: bit_vector(0 TO 15);
file datafile: text open write_mode is "removing.txt";
variable l: line;

begin
 if (mark1 = -1) then
   --/ receiving the data /--
   syn_receive(in_unsigned, t_temp, 99999 ns);
   s.low_count:= t_temp.color.data1;
   syn_receive(in_unsigned, t_temp, 99999 ns);
   s.high_count:= t_temp.color.data1;
   syn_receive(in_unsigned, t_temp, 99999 ns);
   s.scale:= t_temp.color.data1;
   syn_receive(in_unsigned, t_temp, 99999 ns);
   high:= t_temp.color.data1;
   syn_receive(in_unsigned, t_temp, 99999 ns);
   low:= t_temp.color.data1;
   syn_receive(in_unsigned, t_temp, 99999 ns);
   code:= t_temp.color.data1;

   --/ initianising the process /--
   range1:= INTEGER(high - low) + 1;
   high:= low + unsigned_int((range1 * INTEGER(s.high_count))
        / INTEGER(s.scale) - 1);
   low:=  low + unsigned_int((range1 * INTEGER(s.low_count))
        / INTEGER(s.scale));

   --/ converting into bit_vectors /--
   bv16_high := unsignint_to_bv(high,16);
   bv16_low  := unsignint_to_bv(low, 16);
   bv16_code := unsignint_to_bv(code,16);
   mark1:= 0;
 end if;

  --/ starting to remove /--
 if (bv16_high(0) = bv16_low(0)) then
   NULL;
 elsif ((bv16_low(1) = '1') AND (bv16_high(1) = '0')) then
       bv16_code:= bv16_code XOR X"4000";
       bv16_low:=  bv16_low  AND X"3FFF";
       bv16_high:= bv16_high OR  X"4000";
 else
       mark1:= -1;
 end if;
 if (mark1 = 0) then
    bv16_low:=   bv16_low  sll 1;
    bv16_high:= (bv16_high sll 1) OR X"0001";
    bv16_code:=  bv16_code sll 1;
    syn_receive(in_bits, t_temp, 99999 ns);
    bit_tmp:= t_temp.color.data2;
    bv16_code(15):= bv16_code(15) OR bit_tmp;
 end if;

 if (mark1 = -1) then
```

```
    --/ recovering back into unsigneds /--
    high:= bv_to_unsignint(bv16_high);
    low := bv_to_unsignint(bv16_low);
    code:= bv_to_unsignint(bv16_code);

    --/ sending back data /--
    t_temp.color.data1:= high;
    t_temp.color.condi:= true;
    syn_transmit(out_unsigned, t_temp, 99999 ns);
    t_temp.color.data1:= low;
    t_temp.color.condi:= true;
    syn_transmit(out_unsigned, t_temp, 99999 ns);
    t_temp.color.data1:= code;
    t_temp.color.condi:= true;
    syn_transmit(out_unsigned, t_temp, 99999 ns);
  end if;

  --/ counting invoking time /--
  write(l, t_times, right, 1);
  t_times:= not t_times;
  if t_count >= 29 then
     write(l, NOW, right, 15);
     writeline(datafile,l);
     t_count:= -1;
  end if;
  t_count:= t_count + 1;

end process remover;

end behave_remover;
```

- Data_sink

```
USE std.textio.ALL;
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
USE WORK.token_definition.ALL;
USE WORK.token_passing.ALL;
USE WORK.par_vhdl_conversion.ALL;
USE WORK.arith_pack.ALL;

entity data_sink is
      port(in_text: inout token);
end data_sink;

architecture behave_data_sink of data_sink is

signal out_text: token_res;

begin

buf:process
variable que_point: queptr_tk:= creat_que_tk;
begin
   asyn_buffer(in_text, out_text, que_point, 99999 ns);
   wait on in_text, out_text;
end process buf;

store: process
variable c: character;
variable c1: integer;
```

```
variable t_temp: token;
variable code: unsigned_int;
file datastore: text open write_mode is "test_received.txt";
variable l_line: line;

begin
  --/ receiving text /--
  syn_receive(out_text, t_temp, 99999 ns);
  code:= t_temp.color.data1;
  c1:=  integer(code);
  c:=   CHARACTER'Val(c1);
  if (c = '#') then
     wait;
  end if;
  if (c = CR) then
     writeline(datastore, l_line);
  else
     write(l_line, c, right, 1);
  end if;

end process store;

end behave_data_sink;
```

2. Simulation Results

- **Original Message Compressed in Arithmetic Encoding**

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbb
cc
d
wxyz !
#
```

(The character '#' is used as a terminator.)

- **Expanded Message in Arithmetic Decoding**

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbb
cc
d
wxyz !
```

- **Communications**

1. source_contrl

(limited space, omitted therefore)

2. contrl_buildr

(limited space, omitted therefore)

3. contrl_expandr

0111111100110000 512 ns

4. contrl_rmover

1101111011010100 514 ns
0010000100100101 516 ns
1101010000110011 518 ns
1001000011100011 520 ns
1101111011010011 522 ns
1001010111111010 524 ns
0011000100101101 526 ns
1001100100100100 528 ns
0101001000111001 530 ns
0010010110000101 532 ns
1110011101010110 534 ns
1111000000000000 536 ns
0000000000000000 538 ns

5. expandr_rmover

(limited space, omitted therefore)

6. rmover_expandr

(limited space, omitted therefore)

7. buildr_expandr

(limited space, omitted therefore)

8. expandr_sink

(limited space, omitted therefore)

• Process Invocations

1. Control_block

```
01010101010101010101010101010101          58 ns
01010101010101010101010101010101         118 ns
01010101010101010101010101010101         178 ns
01010101010101010101010101010101         238 ns
01010101010101010101010101010101         298 ns
01010101010101010101010101010101         358 ns
01010101010101010101010101010101         418 ns
01010101010101010101010101010101         478 ns
01010101010101010101010101010101         538 ns
```

2. Model_Building

```
01010101010101010101010101010101          58 ns
01010101010101010101010101010101         118 ns
01010101010101010101010101010101         178 ns
01010101010101010101010101010101         238 ns
01010101010101010101010101010101         298 ns
01010101010101010101010101010101         358 ns
01010101010101010101010101010101         418 ns
01010101010101010101010101010101         478 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
```

3. Expander

```
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         510 ns
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         516 ns
01010101010101010101010101010101         516 ns
01010101010101010101010101010101         518 ns
01010101010101010101010101010101         518 ns
01010101010101010101010101010101         520 ns
01010101010101010101010101010101         520 ns
01010101010101010101010101010101         522 ns
01010101010101010101010101010101         528 ns
```

4. Remover

```
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         514 ns
01010101010101010101010101010101         516 ns
01010101010101010101010101010101         516 ns
01010101010101010101010101010101         516 ns
01010101010101010101010101010101         518 ns
01010101010101010101010101010101         518 ns
01010101010101010101010101010101         520 ns
01010101010101010101010101010101         520 ns
01010101010101010101010101010101         520 ns
01010101010101010101010101010101         522 ns
01010101010101010101010101010101         524 ns
01010101010101010101010101010101         526 ns
01010101010101010101010101010101         530 ns
01010101010101010101010101010101         534 ns
01010101010101010101010101010101         536 ns
01010101010101010101010101010101         538 ns
```

Appendix E

1. Synchronous Channel's Gateway on Same Bus

```
USE WORK.ESSENTIAL_DEFINITIONS.ALL;

ENTITY synchro_same IS
     PORT (clk: IN BIT;

        -- address_bus signals --
          atb: IN word;
        -- data_bus signals --
          dtb: INOUT or_lword_res BUS;
        -- control_bus signals --
          as, ds, rw, sd, sa: IN BIT;
          dtack, ready: INOUT and_bit_res BUS := '1'
         );
END synchro_same;

ARCHITECTURE behave_synchro_same OF synchro_same IS
BEGIN
  PROCESS
  VARIABLE mem: mem_lword:= (others => l_zero);
  VARIABLE remarks: mem_bit:= (others => '0');
  VARIABLE tmpi: unsigned_int;
  BEGIN
  -- bus signals setup --
    dtb   <= NULL;
    dtack <= NULL;
    ready <= NULL;
  -- wait for a synchronous channel activated --
    WAIT UNTIL (as = '0' AND ds = '0' AND sd = '0' AND
              sa = '0' AND clk'EVENT AND clk = '1');
  -- synchronous communications on the same bus --
    tmpi:= bv_to_unsignint(atb);
    IF (rw = '1') AND (remarks(tmpi) = '1') THEN
       -- the data is available --
         dtb <= mem(tmpi);
         mem(tmpi):= l_zero;
         remarks(tmpi):= '0';
         ready <= '0';
    ELSIF (rw = '1') AND (remarks(tmpi) = '0') THEN
       -- the data is not available --
         remarks(tmpi):= '1';
         ready <= '1';
    ELSIF (rw = '0') AND (remarks(tmpi) = '0') THEN
       -- the data is not on demand --
         mem(tmpi):= dtb;
         remarks(tmpi):= '1';
         ready <= '1';
    ELSIF (rw = '0') AND (remarks(tmpi) = '1') THEN
       -- the data is on demand --
         remarks(tmpi):= '0';
         ready <= '0';
    END IF;
    dtack <= '0';
    WAIT UNTIL ds = '1';
    dtack <= '1';
```

```
 END PROCESS;

END behave_synchro_same;
```

2. Synchronous Channel's Gateway on Different Buses

```
USE WORK.ESSENTIAL_DEFINITIONS.ALL;

ENTITY synchro_differ IS
     GENERIC (arbitra_id1: word;
           arbitra_id2: word);
     PORT    (clk: IN BIT;

          -- No.1 bus signals --
          -- address bus
            atb1: INOUT or_word_res BUS;
            segmt1: INOUT or_word_res BUS;
          -- data bus
            dtb1: INOUT or_lword_res BUS;
          -- control bus
            as1, ds1: INOUT and_bit_res BUS;
            rw1, sd1, sa1: INOUT or_bit_res BUS;
            dtack1, ready1: INOUT and_bit_res BUS := '1';
          -- arbitration bus
            br1: INOUT and_word_res BUS:= word_high;
            bg1: IN word;
            bbsy1: INOUT and_bit_res BUS:= '1';

          -- No.2 bus signals --
          -- address bus
            atb2: INOUT or_word_res BUS;
            segmt2: INOUT or_word_res BUS;
          -- data bus
            dtb2: INOUT or_lword_res BUS;
          -- control bus
            as2, ds2: INOUT and_bit_res BUS;
            rw2, sd2, sa2: INOUT or_bit_res BUS;
            dtack2, ready2: INOUT and_bit_res BUS := '1';
          -- arbitration bus
            br2: INOUT and_word_res BUS:= word_high;
            bg2: IN word;
            bbsy2: INOUT and_bit_res BUS:= '1'
           );
END synchro_differ;

ARCHITECTURE behave_synchro_differ OF synchro_differ IS

-- the synchronous channels --
  SHARED VARIABLE mem:  mem_lword:= (others => l_zero);
  SHARED VARIABLE remarks: mem_bit:= (others => '0');

-- the queue in inteface1 --
-- //// the queue does not consider overflow !! //// --
  SHARED VARIABLE mem1: mem_lword:= (others => l_zero);
  SHARED VARIABLE remark1_atb: mem_word:= (others => word_zero);
  SHARED VARIABLE remark1_sa: mem_bit:= (others => '0');
  SHARED VARIABLE remark1_sd: mem_bit:= (others => '0');
  SHARED VARIABLE head1, tail1: INTEGER:= 0;
```

```
-- the queue in interface2 --
-- //// the queue does not consider overflow !! //// --
  SHARED VARIABLE mem2: mem_lword:= (others => l_zero);
  SHARED VARIABLE remark2_atb: mem_word:= (others => word_zero);
  SHARED VARIABLE remark2_sa: mem_bit:= (others => '0');
  SHARED VARIABLE remark2_sd: mem_bit:= (others => '0');
  SHARED VARIABLE head2, tail2: INTEGER:= 0;

-- singals for exclusive accesses to shared memories --
  SIGNAL proceed_mem:  permit_res;
  SIGNAL proceed_que1: permit_res;
  SIGNAL proceed_que2: permit_res;

-- signals to synchronize the bus requests --
  SIGNAL dwb1, dwb2, dgb1, dgb2: BIT:= '0';

BEGIN

gateway1: PROCESS
 VARIABLE tmpi: unsigned_int;
 BEGIN
-- No.1 bus signals setup --
  dtb1   <= NULL;
  dtack1 <= NULL;
  ready1 <= NULL;
-- wait for a synchronous channel activated --
  WAIT UNTIL (as1 = '0' AND ds1 = '0' AND sd1 = '1' AND
          sa1 = '0' AND clk'EVENT AND clk = '1');
  -- No. 1 Bus event --
  IF (segmt1 /= arbitra_id1) THEN
    tmpi:= bv_to_unsignint(atb1);
   -- applying for operating on the critical section --
    proceed_mem <= (1, NOW);
    WAIT ON proceed_mem UNTIL (proceed_mem.idnumber = 1);
   -- enter the critical section --
    IF (rw1 = '1') AND (remarks(tmpi) = '1') THEN
      -- the data is available --
        dtb1 <= mem(tmpi);
        mem(tmpi):= l_zero;
        remarks(tmpi):= '0';
      -- releasing the critical section --
        proceed_mem <= (0, NOW);

      -- informing the countpart waiting at bus2 --
      -- applying for operating on the critical section --
        proceed_que2 <= (1, NOW);
        WAIT ON proceed_que2 UNTIL (proceed_que2.idnumber = 1);
        remark2_atb(tail2):= atb1;
        remark2_sd(tail2):= sd1;
        remark2_sa(tail2):= sa1;
        tail2:= tail2 + 1;
        IF tail2 > mem_size THEN
          tail2:= 0;
        END IF;
      -- releasing the critical section --
        proceed_que2 <= (0, NOW);
        ready1 <= '0';
    ELSIF (rw1 = '1') AND (remarks(tmpi) = '0') THEN
      -- the data is not available --
        remarks(tmpi):= '1';
```

```
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
          ready1 <= '1';
      ELSIF (rw1 = '0') AND (remarks(tmpi) = '0') THEN
        -- the data is not on demand --
          mem(tmpi):= dtb1;
          remarks(tmpi):= '1';
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
          ready1 <= '1';
      ELSIF (rw1 = '0') AND (remarks(tmpi) = '1') THEN
        -- the data's been on demand --
          remarks(tmpi):= '0';
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
        -- informing the counterpart at bus2 --
        -- applying for operating on the critical section --
          proceed_que2 <= (1, NOW);
          WAIT ON proceed_que2 UNTIL (proceed_que2.idnumber = 1);
          mem2(tail2):= dtb1;
          remark2_atb(tail2):= atb1;
          remark2_sd(tail2):= sd1;
          remark2_sa(tail2):= sa1;
          tail2:= tail2 + 1;
          IF tail2 > mem_size THEN
            tail2:= 0;
          END IF;
        -- releasing the critical section --
          proceed_que2 <= (0, NOW);
          ready1 <= '0';
      END IF;
    END IF;
    dtack1 <= '0';
    WAIT UNTIL ds1 = '1';
    dtack1 <= '1';
  END PROCESS;

gateway2: PROCESS
  VARIABLE tmpi: unsigned_int;
  BEGIN
 -- No.2 bus signals setup --
    dtb2   <= NULL;
    dtack2 <= NULL;
    ready2 <= NULL;
 -- wait for a synchronous channel activated --
    WAIT UNTIL (as2 = '0' AND ds2 = '0' AND sd2 = '1' AND
            sa2 = '0' AND clk'EVENT AND clk = '1');
    -- No.2 Bus event --
    IF (segmt2 /= arbitra_id2) THEN
      tmpi:= bv_to_unsignint(atb2);
     -- applying for operating on the critical section --
      proceed_mem <= (2, NOW);
      WAIT ON proceed_mem UNTIL (proceed_mem.idnumber = 2);
     -- enter the critical section --
      IF (rw2 = '1') AND (remarks(tmpi) = '1') THEN
        -- the data is available --
          dtb2 <= mem(tmpi);
          mem(tmpi):= l_zero;
          remarks(tmpi):= '0';
        -- releasing the critical section --
```

```
          proceed_mem <= (0, NOW);

        -- informing the countpart waiting at bus1 --
        -- applying for operating on the critical section --
          proceed_que1 <= (2, NOW);
          WAIT ON proceed_que1 UNTIL (proceed_que1.idnumber = 2);
          remark1_atb(tail1):= atb2;
          remark1_sd(tail1):= sd2;
          remark1_sa(tail1):= sa2;
          tail1:= tail1 + 1;
          IF tail1 > mem_size THEN
            tail1:= 0;
          END IF;
        -- releasing the critical section --
          proceed_que1 <= (0, NOW);
          ready2 <= '0';
      ELSIF (rw2 = '1') AND (remarks(tmpi) = '0') THEN
        -- the data is not available --
          remarks(tmpi):= '1';
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
          ready2 <= '1';
      ELSIF (rw2 = '0') AND (remarks(tmpi) = '0') THEN
        -- the data is not on demand --
          mem(tmpi):= dtb2;
          remarks(tmpi):= '1';
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
          ready2 <= '1';
      ELSIF (rw2 = '0') AND (remarks(tmpi) = '1') THEN
        -- the data's been on demand --
          remarks(tmpi):= '0';
        -- releasing the critical section --
          proceed_mem <= (0, NOW);
        -- informing the counterpart at bus1 --
        -- applying for operating on the critical section --
          proceed_que1 <= (2, NOW);
          WAIT ON proceed_que1 UNTIL (proceed_que1.idnumber = 2);
          mem1(tail1):= dtb2;
          remark1_atb(tail1):= atb2;
          remark1_sd(tail1):= sd2;
          remark1_sa(tail1):= sa2;
          tail1:= tail1 + 1;
          IF tail1 > mem_size THEN
            tail1:= 0;
          END IF;
        -- releasing the critical section --
          proceed_que1 <= (0, NOW);
          ready2 <= '0';
      END IF;
    END IF;
    dtack2 <= '0';
    WAIT UNTIL ds2 = '1';
    dtack2 <= '1';
  END PROCESS;

bus_request1: PROCESS
  BEGIN
    br1  <= NULL;
    WAIT UNTIL (dwb1 = '1');
```

```
    br1   <= arbitra_id1;
    WAIT UNTIL (bg1 = arbitra_id1);
    bbsy1 <= '0';
    dgb1  <= '1';
    br1   <= NULL;
    WAIT UNTIL (bg1 = word_high AND dwb1 = '0');
    bbsy1 <= '1';
    dgb1  <= '0';
  END PROCESS;

bus_request2: PROCESS
  BEGIN
    br2   <= NULL;
    WAIT UNTIL (dwb2 = '1');
    br2   <= arbitra_id2;
    WAIT UNTIL (bg2 = arbitra_id2);
    bbsy2 <= '0';
    dgb2  <= '1';
    br2   <= NULL;
    WAIT UNTIL (bg2 = word_high AND dwb2 = '0');
    bbsy2 <= '1';
    dgb2  <= '0';
  END PROCESS;

interface_1: PROCESS
VARIABLE tmp_dtb: l_word;
VARIABLE tmp_atb: word;
VARIABLE tmp_sd, tmp_sa: BIT;
  BEGIN
    atb1  <= NULL;
    segmt1 <= NULL;
    dtb1  <= NULL;
    rw1   <= NULL;
    sd1   <= NULL;
    sa1   <= NULL;
    as1   <= NULL;
    ds1   <= NULL;
  -- applying for operating on the critical section --
    proceed_que1 <= (1, NOW);
    WAIT ON proceed_que1 UNTIL (proceed_que1.idnumber = 1);
    IF head1 /= tail1 THEN
      tmp_dtb:= mem1(head1);
      tmp_atb:= remark1_atb(head1);
      tmp_sd:=  remark1_sd(head1);
      tmp_sa:=  remark1_sa(head1);
      head1:= head1 + 1;
      IF head1 > mem_size THEN
        head1:= 0;
      END IF;
    -- releasing the critical section --
      proceed_que1 <= (0, NOW);
    -- sending the message to the monitor --
      dwb1  <= '1';
      WAIT UNTIL (dgb1 = '1');
      atb1  <= tmp_atb;
      segmt1 <= arbitra_id1;
      dtb1  <= tmp_dtb;
      sd1   <= tmp_sd;
      sa1   <= tmp_sa;
      as1   <= '0';
```

```
      ds1   <= '0';
      WAIT UNTIL dtack1 = '0';
      as1   <= '1';
      ds1   <= '1';
      dwb1  <= '0';
    END IF;
  END PROCESS;

interface_2: PROCESS
VARIABLE tmp_dtb: l_word;
VARIABLE tmp_atb: word;
VARIABLE tmp_sd, tmp_sa: BIT;
  BEGIN
    atb2  <= NULL;
    segmt2 <= NULL;
    dtb2  <= NULL;
    rw2   <= NULL;
    sd2   <= NULL;
    sa2   <= NULL;
    as2   <= NULL;
    ds2   <= NULL;
 -- applying for operating on the critical section --
    proceed_que2 <= (2, NOW);
    WAIT ON proceed_que2 UNTIL (proceed_que2.idnumber = 2);
    IF head2 /= tail2 THEN
      tmp_dtb:= mem2(head2);
      tmp_atb:= remark2_atb(head2);
      tmp_sd:=  remark2_sd(head2);
      tmp_sa:=  remark2_sa(head2);
      head2:=   head2 + 1;
      IF head2 > mem_size THEN
        head2:= 0;
      END IF;
    -- releasing the critical section --
      proceed_que2 <= (0, NOW);

    -- sending the message to the monitor --
      dwb2  <= '1';
      WAIT UNTIL (dgb2 = '1');
      atb2  <= tmp_atb;
      segmt2 <= arbitra_id2;
      dtb2  <= tmp_dtb;
      sd2   <= tmp_sd;
      sa2   <= tmp_sa;
      as2   <= '0';
      ds2   <= '0';
      WAIT UNTIL dtack2 = '0';
      as2   <= '1';
      ds2   <= '1';
      dwb2  <= '0';
    END IF;
  END PROCESS;

END behave_synchro_differ;
```

3. Asynchronous Channel's Gateway on Same Bus

```
USE WORK.ESSENTIAL_DEFINITIONS.ALL;
```

```
ENTITY asynchro_same IS
     GENERIC (que_id: word);
     PORT    (clk: IN BIT;

          -- address_bus signals --
            atb: IN word;
            segmt: IN word;
          -- data_bus signals --
            dtb: INOUT or_lword_res BUS;
          -- control_bus signals --
            as, ds, rw, sd, sa: IN BIT;
            dtack, ready: INOUT and_bit_res BUS := '1'
           );
END asynchro_same;

ARCHITECTURE behave_asynchro_same OF asynchro_same IS
BEGIN
 PROCESS
 VARIABLE tmp_i: INTEGER;
 VARIABLE tmp_que_mem: mem_of_que;
 VARIABLE que_point: queptr:= creat_que;
 VARIABLE mark: BIT:= '0'; -- not visited --
 BEGIN
  dtb   <= NULL;
  dtack <= NULL;
  ready <= NULL;
  WAIT UNTIL (as = '0' AND ds = '0' AND sd = '0' AND
           sa = '1' AND atb = que_id AND
           clk'EVENT AND clk = '1');
-- this asynchronous channel is activated --
  number_of_que(que_point, tmp_i);
  IF (rw = '1') AND (tmp_i = 0) THEN
     -- queue is empty & set up the mark --
       mark:= '1';
       ready <= '1';
  ELSIF (rw = '1') AND (tmp_i > 0) THEN
     -- queue is not empty & data is ready --
       out_que(que_point, tmp_que_mem);
       dtb <= tmp_que_mem.dtb;
       ready <= '0';
  ELSIF (rw = '0') AND (mark = '1') THEN
     -- deleting the marking --
       mark:= '0';
       ready <= '0';
  ELSIF (rw = '0') AND (mark = '0') THEN
     -- the data goes into queue--
       tmp_que_mem.dtb:= dtb;
       in_que(que_point, tmp_que_mem);
       ready <= '0';
  END IF;
  dtack <= '0';
  WAIT UNTIL ds = '1';
  dtack <= '1';
 END PROCESS;

END behave_asynchro_same;
```

4. Asynchronous Channel's Gateway on Different Buses

```
USE WORK.ESSENTIAL_DEFINITIONS.ALL;

ENTITY asynchro_differ IS
     GENERIC (arbitra_id: word; que_id: word);
     PORT    (clk: IN BIT;

          -- No.1 bus signals --
          -- address bus --
            atb1: IN word;
            segmt1: IN word;
          -- data bus --
            dtb1: IN l_word;
          -- control bus --
            as1, ds1: IN BIT;
            rw1, sd1, sa1: IN BIT;
            dtack1, ready1: INOUT and_bit_res BUS := '1';

          -- No.2 bus signals --
          -- address bus --
            atb2: INOUT or_word_res BUS;
            segmt2: INOUT or_word_res BUS;
          -- data bus --
            dtb2: INOUT or_lword_res BUS;
          -- control bus --
            as2, ds2: INOUT and_bit_res BUS;
            rw2: IN BIT;
            sd2, sa2: INOUT or_bit_res BUS;
            dtack2, ready2: INOUT and_bit_res BUS := '1';
          -- arbitration bus --
            br2: INOUT and_word_res BUS:= word_high;
            bg2: IN word;
            bbsy2: INOUT and_bit_res BUS:= '1'
           );
END asynchro_differ;

ARCHITECTURE behave_asynchro_differ OF asynchro_differ IS

-- //// the queue does not consider overflows !! //// --
-- the main queue for an asynchronous channel --
  SHARED VARIABLE mem:  mem_lword:= (others => l_zero);
  SHARED VARIABLE mark: BIT:= '0'; -- not on demand --
  SHARED VARIABLE head, tail: INTEGER:= 0;

-- the queue in the bus inteface --
-- //// the queue does not consider overflows !! //// --
  SHARED VARIABLE mem_int_face: mem_lword:= (others => l_zero);
  SHARED VARIABLE mark_atb: mem_word:= (others => word_zero);
  SHARED VARIABLE mark_sa: mem_bit:= (others => '0');
  SHARED VARIABLE mark_sd: mem_bit:= (others => '0');
  SHARED VARIABLE head_int_face, tail_int_face: INTEGER:= 0;

-- signals to synchronize the bus requests --
  SIGNAL dwb, dgb: BIT:= '0';

-- singals for exclusive accesses to shared memories --
  SIGNAL proceed_mem:     permit_res;
  SIGNAL proceed_mem_int: permit_res;
```

```
BEGIN
asynch_gateway1: PROCESS
  BEGIN
 -- No.1 bus signals setup --
   dtack1 <= NULL;
   ready1 <= NULL;
   WAIT UNTIL (as1 = '0' AND ds1 = '0' AND sd1 = '1' AND
            sa1 = '1' AND atb1 = que_id AND
            clk'EVENT AND clk = '1');
 -- this asynchronous channel is activated --
 -- applying for operating on the critical section --
   proceed_mem <= (1, NOW);
   WAIT ON proceed_mem UNTIL proceed_mem.idnumber = 1;
   IF mark = '0' THEN -- the data is not on demand --
   -- enter the critical section --
     mem(tail):= dtb1;
     tail:= tail + 1;
     IF tail > mem_size THEN
       tail:= 0;
     END IF;
   -- releasing the critical section --
     proceed_mem <= (0, NOW);
   ELSE
   -- deleting the marking --
     mark:= '0';
   -- releasing the critical section --
     proceed_mem <= (0, NOW);
   -- applying for operating on the bus interface queue --
     proceed_mem_int <= (1, NOW);
     WAIT ON proceed_mem_int UNTIL (proceed_mem_int.idnumber = 1);
     mem_int_face(tail_int_face):= dtb1;
     mark_atb(tail_int_face):= atb1;
     mark_sa(tail_int_face):=  sa1;
     mark_sd(tail_int_face):=  sd1;
     tail_int_face:= tail_int_face + 1;
     IF tail_int_face > mem_size THEN
       tail_int_face:= 0;
     END IF;
   -- releasing the critical section --
     proceed_mem_int <= (0, NOW);
   END IF;
   ready1 <= '0';
   dtack1 <= '0';
   WAIT UNTIL ds1 = '1';
   dtack1 <= '1';
  END PROCESS;

asynch_gateway2: PROCESS
  BEGIN
 -- No.2 bus signals setup --
   dtb2   <= NULL;
   dtack2 <= NULL;
   ready2 <= NULL;
   WAIT UNTIL (as2 = '0' AND ds2 = '0' AND sd2 = '1' AND
            sa2 = '1' AND atb2 = que_id AND
            clk'EVENT AND clk = '1');
 -- applying for operating on the critical section --
   proceed_mem <= (2, NOW);
   WAIT ON proceed_mem UNTIL proceed_mem.idnumber = 2;
 -- enter the critical section --
```

```
    IF (segmt2 /= arbitra_id) THEN
       IF head /= tail THEN -- the data is available --
         dtb2 <= mem(head);
         head := head + 1;
         IF head > mem_size THEN
           head:= 0;
         END IF;
         ready2 <= '0';
       ELSE
         mark:= '1';
         ready2 <= '1';
       END IF;
    END IF;
  -- releasing the critical section --
    proceed_mem <= (0, NOW);
    dtack2 <= '0';
    WAIT UNTIL ds2 = '1';
    dtack2 <= '1';
   END PROCESS;

bus_request: PROCESS
  BEGIN
    WAIT UNTIL (dwb = '1');
    br2  <= arbitra_id;
    WAIT UNTIL (bg2 = arbitra_id);
    bbsy2 <= '0';
    dgb  <= '1';
    br2  <= NULL;
    WAIT UNTIL (bg2 = word_high AND dwb = '0');
    bbsy2 <= '1' ;
    dgb  <= '0';
  END PROCESS;

asyn_int_face: PROCESS
VARIABLE tmp_dtb: l_word;
VARIABLE tmp_atb: word;
VARIABLE tmp_sd, tmp_sa: BIT;
  BEGIN
    atb2  <= NULL;
    dtb2  <= NULL;
    segmt2 <= NULL;
    sa2   <= NULL;
    sd2   <= NULL;
    as2   <= NULL;
    ds2   <= NULL;
    br2   <= NULL;
  -- applying for operating on the queue of bus interface --
    proceed_mem_int <= (2, NOW);
    WAIT ON proceed_mem_int UNTIL (proceed_mem_int.idnumber = 2);
    IF head_int_face /= tail_int_face THEN
       tmp_dtb:= mem_int_face(head_int_face);
       tmp_atb:= mark_atb(head_int_face);
       tmp_sa:=  mark_sa(head_int_face);
       tmp_sd:=  mark_sd(head_int_face);
       head_int_face:= head_int_face + 1;
       IF head_int_face > mem_size THEN
         head_int_face:= 0;
       END IF;
    -- releasing the critical section --
       proceed_mem_int <= (0, NOW);
```

```
    dwb   <= '1';
    WAIT UNTIL (dgb = '1');
    dtb2  <= tmp_dtb;
    atb2  <= tmp_atb;
    segmt2 <= arbitra_id;
    sa2   <= tmp_sa;
    sd2   <= tmp_sd;
    as2   <= '0';
    ds2   <= '0';
    WAIT UNTIL dtack2 = '0';
    as2   <= '1';
    ds2   <= '1';
    dwb   <= '0';
  END IF;
 END PROCESS;

END behave_asynchro_differ;
```

Appendix F

1. The Top-Level Co-BSL Specification

```
-----------------------------------------------------------------------------------------------------------------------
-- The Co-BSL specification of the top-level process graph illustrated in Figure 6.4 is listed
-- below. Although some of program details are not included, the program has adequately
-- demonstrated the usage of Co-BSL as a specification tool in the codesign methodology.
-----------------------------------------------------------------------------------------------------------------------

CODESIGN RDSC

-- The top-level Co-BSL descriptions for RDC System

PATH
t_r   WIRE  BIT;
r_d   SYNC  INT;
d_d   ASYN  INT;

PRIMITIVE

#include "co.prims"

EXTERNAL

#include "co.externs"

EXCUTION
r:   R_P;
d:   D_P;
tt:  Transmitters;
ds:  Data_Storage;

CONNECT_PATHS
t_r:   FROM tt.outp1 TO r.inp1;
r_d:   FROM r.outp1  TO d.inp1;
d_d:   FROM d.outp1 TO ds.inp1;

END_CODESIGN

-- "co.prims"
-- primitive process descriptions

PROCESS D_P OF CONTROL_PROCESS
INPORTS
inp1 SYNC INT;

OUTPORTS
outp1 ASYN  INT;

VARIABLES
     -- variable declaration

BEGIN
     -- sequential code is to be added in subsequent refinements
END
```

```
END_PROCESS

PROCESS R_P OF FUNCTION_SERVER
INPORTS
inp1 WIRE BIT;

OUTPORTS
outp1 SYNC INT;

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code is to be added in subsequent refinements

END

END_PROCESS

-- "co.externs"
-- external interface objects

INTERFACE Transmitters
OUTPORTS
outp1 WIRE BIT;

END_INTERFACE

INTERFACE Data_Storage
INPORTS
Inp1 ASYN  INT;

END_INTERFACE
```

2. The Low-Level Co-BSL Specification

```
-----------------------------------------------------------------------------------------------------------------------

-- The Co-BSL specification of the low-level process graph illustrated in Figure 6.7 is listed
-- below. Although some of program details are not included, the program has adequately
-- demonstrated the usage of Co-BSL as a specification tool in the codesign methodology.
-----------------------------------------------------------------------------------------------------------------------

CODESIGN RDSC

-- The low-level Co-BSL descriptions for RDC System

PATH
trsmit_pm_1   WIRE   BIT;
trsmit_pm_2   WIRE   BIT;
trsmit_pcw_1  WIRE   BIT;
trsmit_pcw_2  WIRE   BIT;
pm_corrtr_1   SYNC   ARRAY [16] BIT;
pm_corrtr_2   SYNC   ARRAY [16] BIT;
pcw_corrtr_1  SYNC  ARRAY [10] BIT;
pcw_corrtr_2  SYNC  ARRAY [10] BIT;
corrtr_pcw_1  SYNC  ARRAY [10] BIT;
corrtr_pcw_1  SYNC  ARRAY [10] BIT;
corrtr_f_1      SYNC  INT;
corrtr_f_2      SYNC  INT;
```

```
f_1_contl       SYNC  INT;
f_2_contl       SYNC  INT;
contl_str       ASYN  INT;

PRIMITIVE

#include "co.prims"

CLASSES

#include "co.clases"

EXTERNAL

#include "co.externs"

EXCUTION
pm_a:    PM_16;
pm_b:    PM_16;
pcw_a:   PCW_10;
pcw_b:   PCW_10;
c_a:     Corrector;
c_b:     Corrector;
f1:      F_A;
f2:      F_B;
c:       Control_Matrix;
t1:      Transmitter_A;
t2:      Transmitter_B;
d:       Data_Storage;

CONNECT_PATHS
trsmit_pm_1:  FROM t1.outp1 TO pm_a.inp1;
trsmit_pm_2:  FROM t2.outp1 TO pm_b.inp1;
trsmit_pcw_1: FROM t1.outp2 TO pcw_a.inp2;
trsmit_pcw_2: FROM t2.outp2 TO pcw_b.inp2;
pm_corrtr_1:  FROM pm_a.outp1 TO c_a.inp1;
pm_corrtr_2:  FROM pm_b.outp1 TO c_b.inp1;
pcw_corrtr_1: FROM pcw_a.outp1 TO c_a.inp2;
pcw_corrtr_2: FROM pcw_b.outp1 TO c_b.inp2;
corrtr_pcw_1: FROM c_a.outp1 TO pcw_a.inp1;
corrtr_pcw_2: FROM c_b.outp1 TO pcw_b.inp1;
corrtr_f_1:   FROM c_a.outp2 TO f1.inp1;
corrtr_f_2:   FROM c_b.outp2 TO f2.inp1;
f_1_contl:    FROM f1.outp1 TO c.inp1;
f_2_contl:    FROM f2.outp1 TO c.inp2;
contl_str:    FROM c.outp1 TO d.inp1;

END_CODESIGN

-- "co.prims"
-- primitive process descriptions

PROCESS Control_Matrix OF CONTROL_PROCESS

INPORTS
f_1_contl SYNC INT;
f_2_contl SYNC INT;

OUTPORTS
contl_str ASYN INT;
```

```
VARIABLES
tmp: INT;
buffer_c: ARRAY(0 TO 9, 0 TO 4) OF INT;
buffer_a: ARRAY(0 TO 9) OF INT;
buffer_b: ARRAY(0 TO 9) OF INT;
m_b: ARRAY(0 TO 4) OF BIT;

BEGIN
  FOR (i= 0; i < 10; i + 1) LOOP

  --/ input one row from transmiter a /--
     FOR (j= 0; j < 10; j + 1) LOOP
          RECEIVE(f_1_contl, buffer_a[j]);
     END_FOR;

  --/ input one column from transmiter b /--
     FOR (j= 0; j < 5; j + 1) LOOP
          IF m_b(j) = '0' THEN
             FOR (k = 0; k < 10; k + 1) LOOP
                  RECEIVE(f_2_contl, buffer_c[k,j]);
            END_FOR;
            m_b[j] := '1';
          END_IF;

        FOR (k = 0; k < 10; k + 1) LOOP
             buffer_b[k]:= buffer_c[k,j];
        END_FOR;

        tmp:= 0;
        FOR (k = 0; k < 10; k +1) LOOP
             tmp:= buffer_a[k] * buffer_b[k] + tmp;
        END_FOR;

        ASYN-SEND(contl_str, tmp);
     END_FOR;

  END_FOR;

END

END PROCESS


PROCESS pm_16 OF FUNCTION_SERVER

INPORTS
trsmit_pm: BIT;

OUTPORTS
pm_corrtr: ARRAY [26] BIT;

VARIABLES
temp: BIT;
t_tmp: INT;
t_back: BIT;
t_forward: BIT;

data_buffer: ARRAY [26] BIT;
buf_reg: ARRAY [16] BIT;
```

```
syndrom: ARRAY [10] BIT;

BEGIN
   t_tmp:= t_tmp + 1;

   IF (t_tmp < 16) THEN
      RECEIVE(trsmit_pm, temp);

   --/ processing 16 bits information /--
      buf_reg:= buf_reg[1 to 15] & temp;
      t_back:=  syndrom[9];
      t_forward:= temp;
      syndrom:= (t_back XOR t_forward) & (syndrom[0] XOR t_forward) &
                  syndrom[1] & (syndrom[2] XOR t_back XOR t_forward) &
                 (syndrom[3] XOR t_back XOR t_forward) & (syndrom[4]
                  XOR t_back) & syndrom[5] & (syndrom[6] XOR t_back) &
                 (syndrom[7] XOR t_back XOR t_forward) & (syndrom[8]
                  XOR t_forward);

   ELSE
   --/ sending off 16 bits information & 10 bits syndrom /--
      data_buffer(0 to 31):= buf_reg(0 to 15) & syndrom(0 to  9);
      SYN-SEND(pm_corrtr, data_buffer);

   --/ initializing the pm_16 again /--
      t_tmp:= -1;
   END_IF;

END

END_PROCESS


PROCESS PCW_10 OF FUNCTION_SERVER
INPORTS
inp1 SYNC ARRAY [10] BIT;
inp2 WIRE  BIT;

OUTPORTS
outp1 SYNC ARRAY [10] BIT;

CONSTRUCTORS
     -- constructor declaration

VARIABLES
     -- variable declaration

BEGIN
     -- sequential code
END

END_PROCESS

PROCESS Corrector OF FUNCTION_SERVER
INPORTS
inp1 SYNC ARRAY [16] BIT;
inp2 SYNC ARRAY [10] BIT;

OUTPORTS
outp1 SYNC ARRAY [10] BIT;
```

```
outp2 SYNC INT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

PROCESS F_A OF FUNCTION_SERVER
INPORTS
inp1 SYNC INT;

OUTPORTS
outp1 SYNC INT;

CONSTRUCTORS
    -- constructor declaration

VARIABLES
    -- variable declaration

BEGIN
    -- sequential code
END

END_PROCESS

-- "co.clases"
-- decomposable class

D_CLASS Receivers OF FUNCTION_SERVER
INPORTS
inp1 WIRE  BIT;
inp2 WIRE  BIT;

OUTPORTS
Outp1 SYNC INT;

CONSTRUCTORS
    -- constructor declaration

PATH
pm_corrtr  SYNC ARRAY [16] BIT;
corrtr_pcw SYNC ARRAY [10] BIT;
pcw_corrtr SYNC ARRAY [10] BIT;

EXCUTION
p16: PM_16;
p10: PCW_10;
c:   Corrector;

CONNECT_PATHS
pm_corrtr: FROM p16.outp1  TO c.inp1;
corrtr_pcw: FROM c.outp1   TO p10.inp1
```

```
pcw_corrtr: FROM p10.outp1 TO c.inp2

CONNECT_PORTS

END_CLASS

-- "co.externs"
-- external interface objects

INTERFACE Transmitter_A
OUTPORTS
outp1 WIRE BIT;

END_INTERFACE

INTERFACE Transmitter_B
OUTPORTS
outp1 WIRE BIT;

END_INTERFACE

INTERFACE Data_Storage
INPORTS
Inp1 ASYN  INT;

END_INTERFACE
```

Appendix G

1. The VHDL Simulation Program for Verification and Profiling

```
--/ here are definications for VHDL packages and libraries /--

ENTITY top_rdc IS
END top_rdc;

ARCHITECTURE behave_top_rdc OF top_rdc IS

COMPONENT transmit_a
     port(sent_16s: inout token;
          sent_10s: inout token);
end COMPONENT transmit_a;

COMPONENT  transmit_b
     port(sent_16s: inout token;
          sent_10s: inout token);
END COMPONENT transmit_b;

COMPONENT pm_16
     generic (file_name: string);
     port(receiv_16s: inout token;
             out_s: inout token);
END COMPONENT pm_16;

COMPONENT pcw_10
     generic (file_name: string);
     port(receiv_10s: inout token;
          receiv_syndrom: inout token;
          out_s: inout token);
END COMPONENT pcw_10;

COMPONENT corrector
     generic (file_name: string);
     port(in_pm: inout token; in_pcw: inout token;
          out_pcw: inout token; corrected: inout token);
END COMPONENT corrector;

COMPONENT f_a_block
        PORT(in_16:  inout token;
             out_16: inout token);
END COMPONENT f_a_block;

COMPONENT f_b_block
        PORT(in_16:  inout token;
             out_16: inout token);
END COMPONENT f_b_block;

COMPONENT control_block
        PORT(in_f_a: inout token;
             in_f_b: inout token;
       data_storage: inout token);
END COMPONENT control_block;

COMPONENT store
     port(in_text: inout token);
END COMPONENT store;
```

```
SIGNAL trsmit_pm_1, trsmit_pcw_1:  token_res;
SIGNAL trsmit_pm_2, trsmit_pcw_2:  token_res;
SIGNAL pm_corrtr_1, pm_corrtr_2:   token_res;
SIGNAL corrtr_pcw_1, pcw_corrtr_1: token_res;
SIGNAL corrtr_pcw_2, pcw_corrtr_2: token_res;
SIGNAL corrtr_f_1, corrtr_f_2:     token_res;
SIGNAL f_1_contl, f_2_contl, contl_str: token_res;

BEGIN
T1: transmit_a
     port map(trsmit_pm_1, trsmit_pcw_1);

T2: transmit_b
     port map(trsmit_pm_2, trsmit_pcw_2);

T3: pm_16
     generic map ("pm_16_a.txt")
     port map (trsmit_pm_1, pm_corrtr_1);

T4: pm_16
     generic map ("pm_16_b.txt")
     port map (trsmit_pm_2, pm_corrtr_2);

T5: pcw_10
     generic map ("pcw_10_a.txt")
     port map (trsmit_pcw_1, corrtr_pcw_1, pcw_corrtr_1);

T6: pcw_10
     generic map ("pcw_10_b.txt")
     port map (trsmit_pcw_2, corrtr_pcw_2, pcw_corrtr_2);

T7: corrector
     generic map ("corrector_a.txt")
     port map (pm_corrtr_1, pcw_corrtr_1, corrtr_pcw_1, corrtr_f_1);

T8: corrector
     generic map ("corrector_b.txt")
     port map (pm_corrtr_2, pcw_corrtr_2, corrtr_pcw_2, corrtr_f_2);

T9: f_a_block
       port map (corrtr_f_1, f_1_contl);

T10: f_b_block
       port map (corrtr_f_2, f_2_contl);

T11: control_block
       port map (f_1_contl, f_2_contl, contl_str);

T12: store
       port map (contl_str);

R1: process(pm_corrtr_1)
     file datafile: text open write_mode is "pm_corrtr_1.txt";
     variable l: line;
     variable temp: unsigned_int;
     variable counter: integer:= 0;
     variable buf: bit_vector(0 to 15);

     begin
```

```
    if pm_corrtr_1.status = active_source then
      if counter = 0 then
        temp:= pm_corrtr_1.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 15 loop
          write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
        counter:= 1;
      else
        temp:= pm_corrtr_1.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 9 loop
          write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
        counter:= 0;
      end if;
    end if;
  end process;

R2: process(pm_corrtr_2)
    file datafile: text open write_mode is "pm_corrtr_2.txt";
    variable l: line;
    variable temp: unsigned_int;
    variable counter: integer:= 0;
    variable buf: bit_vector(0 to 15);

    begin
     if pm_corrtr_2.status = active_source then
       if counter = 0 then
         temp:= pm_corrtr_2.color.data1;
         buf:= unsignint_to_bv(temp, 16);
         for i in 0 to 15 loop
           write(l, buf(i), right, 1);
         end loop;
         writeline(datafile,l);
         counter:= 1;
       else
         temp:= pm_corrtr_2.color.data1;
         buf:= unsignint_to_bv(temp, 16);
         for i in 0 to 9 loop
           write(l, buf(i), right, 1);
         end loop;
         writeline(datafile,l);
         counter:= 0;
       end if;
     end if;
    end process;


R3: process(corrtr_pcw_1)
    file datafile: text open write_mode is "corrtr_pcw_1.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
     if corrtr_pcw_1.status = active_source then
       temp:= corrtr_pcw_1.color.data1;
```

```
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 9 loop
           write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R4: process(corrtr_pcw_2)
    file datafile: text open write_mode is "corrtr_pcw_2.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if corrtr_pcw_2.status = active_source then
        temp:= corrtr_pcw_2.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 9 loop
           write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R5: process(pcw_corrtr_1)
    file datafile: text open write_mode is "pcw_corrtr_1.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if pcw_corrtr_1.status = active_source then
        temp:= pcw_corrtr_1.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 9 loop
           write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R6: process(pcw_corrtr_2)
    file datafile: text open write_mode is "pcw_corrtr_2.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if pcw_corrtr_2.status = active_source then
        temp:= pcw_corrtr_2.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 9 loop
           write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R7: process(corrtr_f_1)
```

```
    file datafile: text open write_mode is "corrtr_f_1.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if corrtr_f_1.status = active_source then
        temp:= corrtr_f_1.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 15 loop
          write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R8: process(corrtr_f_2)
    file datafile: text open write_mode is "corrtr_f_2.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if corrtr_f_2.status = active_source then
        temp:= corrtr_f_2.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 15 loop
          write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R9: process(f_1_contl)
    file datafile: text open write_mode is "f_1_contl.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if f_1_contl.status = active_source then
        temp:= f_1_contl.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 15 loop
          write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R10: process(f_2_contl)
    file datafile: text open write_mode is "f_2_contl.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if f_2_contl.status = active_source then
        temp:= f_2_contl.color.data1;
        buf:= unsignint_to_bv(temp, 16);
```

```
        for i in 0 to 15 loop
            write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

R11: process(contl_str)
    file datafile: text open write_mode is "contl_str.txt";
    variable l: line;
    variable tmp: integer;
    variable temp: unsigned_int;
    variable buf: bit_vector(0 to 15);
    begin
      if contl_str.status = active_source then
        temp:= contl_str.color.data1;
        buf:= unsignint_to_bv(temp, 16);
        for i in 0 to 15 loop
            write(l, buf(i), right, 1);
        end loop;
        writeline(datafile,l);
      end if;
    end process;

END behave_top_rdc;
```

Appendix H

1. The C Program for Assessment of Software Performance

```
#include <stdio.h>

FILE *spa16;
FILE *spa10;
FILE *spb16;
FILE *spb10;

/* these are in correspondence to the VHDL transmit_a */
unsigned short tmit_a_16(void) {
  char read_in;

  read_in = fgetc(spa16);
  return (read_in ? 1 : 0);

} /* end of tmit_a_16() */

unsigned short tmit_a_10(void) {
  char read_in;

  read_in = fgetc(spa10);
  return (read_in ? 1 : 0);

} /* end of tmit_a_10() */

/* these are in correspondence to the VHDL transmit_b */
unsigned short tmit_b_16(void) {
  char read_in;

  read_in = fgetc(spb16);
  return (read_in ? 1 : 0);

} /* end of tmit_b_16() */

unsigned short tmit_b_10(void) {
  char read_in;

  read_in = fgetc(spb10);
  return (read_in ? 1 : 0);

} /* end of tmit_b_10() */

/* this is in correspondence to VHDL pm_16_a */
void pm_16_a(unsigned short message[], unsigned short syndrom[])
 {
  int i, j;
  unsigned short feed_back, feed_forward;

  /* receiving 16 bits of information */
  for (i = 0; i < 16; i++) {

      feed_forward = message[i] = tmit_a_16();

      feed_back    = syndrom[9];
```

```
    /* a rotation of the syndrom */
    for (j = 9; j > 0; j--)
        syndrom[j] = syndrom[j-1];

    syndrom[9] ^= feed_forward;
    syndrom[8]  = syndrom[8] ^ feed_back ^ feed_forward;
    syndrom[7] ^= feed_back;
    syndrom[5] ^= feed_back;
    syndrom[4]  = syndrom[4] ^ feed_back ^ feed_forward;
    syndrom[3]  = syndrom[3] ^ feed_back ^ feed_forward;
    syndrom[1] ^= feed_forward;
    syndrom[0]  = feed_back ^ feed_forward;

    j = 9;

  } /* end of for loop */

} /* end of pm_16_a() */

/* this is in correspondence to VHDL pcw_10_a */
void pcw_10_a(unsigned short syndrom[])
 {
  int i, j;
  unsigned short feed_back, feed_forward, temp;
  static int tmp_offset = 0;
  static unsigned short offset_a[] =
              {0, 0, 1, 1, 1, 1, 1, 1, 0, 0};
  static unsigned short offset_b[] =
              {0, 1, 1, 0, 0, 1, 1, 0, 0, 0};
  static unsigned short offset_c[] =
              {0, 1, 0, 1, 1, 0, 1, 0, 0, 0};
  static unsigned short offset_d[] =
              {0, 1, 1, 0, 1, 1, 0, 1, 0, 0};

   /* receiving 10 bits of check word */
   for (i = 0; i < 10; i++) {

       temp = tmit_a_10();

       switch (tmp_offset) {
       case 0:
           feed_forward = offset_a[i] ^ temp; break;
       case 1:
           feed_forward = offset_b[i] ^ temp; break;
       case 2:
           feed_forward = offset_c[i] ^ temp; break;
       case 3:
           feed_forward = offset_d[i] ^ temp; break;
       }
      feed_back = syndrom[9];

      /* a rotation of the syndrom */
      for (j = 9; j > 0; j--)
          syndrom[j] = syndrom[j-1];

      syndrom[9] ^= feed_forward;
      syndrom[8]  = syndrom[8] ^ feed_back ^ feed_forward;
      syndrom[7] ^= feed_back;
      syndrom[5] ^= feed_back;
      syndrom[4]  = syndrom[4] ^ feed_back ^ feed_forward;
```

```
    syndrom[3]  = syndrom[3] ^ feed_back ^ feed_forward;
    syndrom[1] ^= feed_forward;
    syndrom[0]  = feed_back ^ feed_forward;

  } /* end of for loop */

  if (tmp_offset++ > 3)
     tmp_offset = 0;

 } /* end of pcw_10_a() */

/* this is in correspondence to VHDL pm_16_b */
void pm_16_b(unsigned short message[], unsigned short syndrom[])
 {
  int i, j;
  unsigned short feed_back, feed_forward;

  /* receiving 16 bits of information */
  for (i = 0; i < 16; i++) {

     feed_forward = message[i] = tmit_b_16();

     feed_back    = syndrom[9];

     /* a rotation of the syndrom */
     for (j = 9; j > 0; j--)
        syndrom[j] = syndrom[j-1];

     syndrom[9] ^= feed_forward;
     syndrom[8]  = syndrom[8] ^ feed_back ^ feed_forward;
     syndrom[7] ^= feed_back;
     syndrom[5] ^= feed_back;
     syndrom[4]  = syndrom[4] ^ feed_back ^ feed_forward;
     syndrom[3]  = syndrom[3] ^ feed_back ^ feed_forward;
     syndrom[1] ^= feed_forward;
     syndrom[0]  = feed_back ^ feed_forward;

  } /* end of for loop */

 } /* end of pm_16_b() */

/* this is in correspondence to VHDL pcw_10_b */
void pcw_10_b(unsigned short syndrom[])
 {
  int i, j;
  unsigned short feed_back, feed_forward, temp;
  static int tmp_offset = 0;
  static unsigned short offset_a[] =
              {0, 0, 1, 1, 1, 1, 1, 1, 0, 0};
  static unsigned short offset_b[] =
              {0, 1, 1, 0, 0, 1, 1, 0, 0, 0};
  static unsigned short offset_c[] =
              {0, 1, 0, 1, 1, 0, 1, 0, 0, 0};
  static unsigned short offset_d[] =
              {0, 1, 1, 0, 1, 1, 0, 1, 0, 0};

  /* receiving 10 bits of check word */
  for (i = 0; i < 10; i++) {

     temp = tmit_b_10();
```

```
        switch (tmp_offset) {
        case 0:
            feed_forward = offset_a[i] ^ temp; break;
        case 1:
            feed_forward = offset_b[i] ^ temp; break;
        case 2:
            feed_forward = offset_c[i] ^ temp; break;
        case 3:
            feed_forward = offset_d[i] ^ temp; break;
        }
       feed_back = syndrom[9];

       /* a rotation of the syndrom */
       for (j = 9; j > 0; j--)
          syndrom[j] = syndrom[j-1];

       syndrom[9] ^= feed_forward;
       syndrom[8]  = syndrom[8] ^ feed_back ^ feed_forward;
       syndrom[7] ^= feed_back;
       syndrom[5] ^= feed_back;
       syndrom[4]  = syndrom[4] ^ feed_back ^ feed_forward;
       syndrom[3]  = syndrom[3] ^ feed_back ^ feed_forward;
       syndrom[1] ^= feed_forward;
       syndrom[0]  = feed_back ^ feed_forward;

    } /* end of for loop */

    if (tmp_offset++ > 3)
       tmp_offset = 0;

  } /* end of pcw_10_b() */

 /* this is in correspondence to VHDL corrector_a */
 unsigned short corrector_a(void)
  {
   int i;
   unsigned short buffer = 0;
   unsigned short m_ssage[16];
   unsigned short s_ndrom[10];

   for (i = 0; i < 16; i++)
       m_ssage[i] = 0;
   for (i = 0; i < 10; i++)
       s_ndrom[i] = 0;

   pm_16_a(m_ssage, s_ndrom);
   pcw_10_a(s_ndrom);

   /* corrections go here */
   for (i = 0; i < 16; i++) {

       buffer <<= 1;

       if (s_ndrom[0] | s_ndrom[1] | s_ndrom[2] | s_ndrom[3] | s_ndrom[4]) {
            if (m_ssage[i] == 0)
               buffer &= 0177776;
            else
               buffer |= 01;
           }
```

```
        else {
            /* corrections are needed */
            if ((s_ndrom[9] ^ m_ssage[i]) != 0)
                buffer |= 01;
            else
                buffer &= 0177776;
        }

    } /* end of for loop */

        return (buffer);

    } /* end of corrector_a() */

/* this is in correspondence to VHDL corrector_b */
unsigned short corrector_b(void)
 {
  int i;
  unsigned short buffer = 0;
  unsigned short m_ssage[16];
  unsigned short s_ndrom[10];

  for (i = 0; i < 16; i++)
      m_ssage[i] = 0;
  for (i = 0; i < 10; i++)
      s_ndrom[i] = 0;

  pm_16_b(m_ssage, s_ndrom);
  pcw_10_b(s_ndrom);

  /* corrections go here */
  for (i = 0; i < 16; i++) {

      buffer <<= 1;

      if (s_ndrom[0] | s_ndrom[1] | s_ndrom[2] | s_ndrom[3] | s_ndrom[4]) {
          if (m_ssage[i] == 0)
              buffer &= 0177776;
          else
              buffer |= 01;
          }
      else {
          /* corrections are needed */
          if ((s_ndrom[9] ^ m_ssage[i]) != 0)
              buffer |= 01;
          else
              buffer &= 0177776;
      }

  } /* end of for loop */

      return (buffer);

  } /* end of corrector_b() */

/* this is in correspondence to VHDL f_a */
unsigned short f_a(void)
 {
  unsigned short a;
```

```
  a = corrector_a();

  a = ((a * a + 1) * (a + 1)) / (a * a + 2);

  return (a);

/*  return (++a); */
 }

/* this is in correspondence to VHDL f_b */
unsigned short f_b(void)
 {
  unsigned short b;

   b = corrector_b();

   b = ((b * b + 1) * (b + 1)) / (b * b + 4);

   return (b);
 }

/* this is in correspondence to VHDL storage */
void storage(int store, FILE *fp)
 {
  static int i = -1;

    if (++i < 5)
      fprintf(fp, "%8u", store);
      else
        { fprintf(fp, "\n%8u", store); i = 0;}

 }

/* this is in correspondence to VHDL control_block */
int main(void)
{
  int i, j, k, sum;
  short int mark_b[]  = {1, 1, 1, 1, 1};
  unsigned short buffer_a[10], buffer_b[10], buffer_c[10][5];
  FILE *sp;

  sp = fopen("store.txt", "w");
  spa16 = fopen("tmita16", "r");
  spa10 = fopen("tmita10", "r");
  spb16 = fopen("tmitb16", "r");
  spb10 = fopen("tmitb10", "r");

  for (i = 0; i < 10; i++) {

      /* input one row from transmiter a */
      for (j = 0; j < 10; j++)
          buffer_a[j] = f_a();

      for (j = 0; j < 5; j++) {

          if (mark_b[j] == 1) {

              /* input one column from transmiter b */
              for (k = 0; k < 10; k++)
                  buffer_c[k][j] = f_b();
```

```
            mark_b[j] = 0;
        }

        for (k = 0; k < 10; k++)
            buffer_b[k] = buffer_c[k][j];

        for (sum = 0, k = 0; k < 10; k++)
            sum += buffer_a[k] * buffer_b[k];

        storage(sum, sp);
    }

  }

  fclose(sp);
  fclose(spa16);
  fclose(spa10);
  fclose(spb16);
  fclose(spb10);

} /* end of main */
```

Appendix I

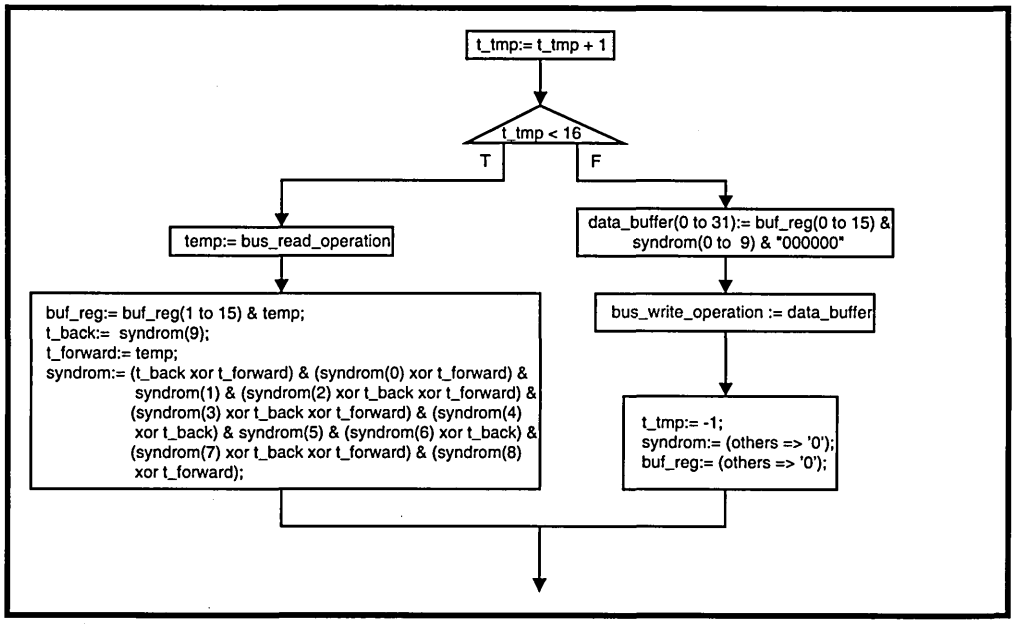
1. PM_16_A/B

- **VHDL Program**

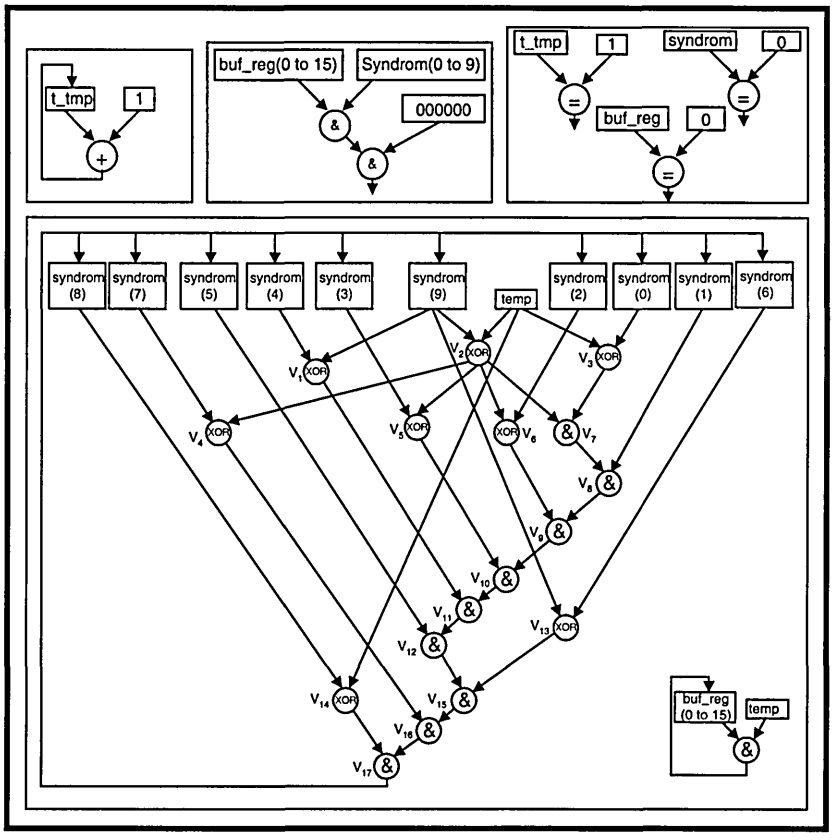
```
---/ here are definitions for VHDL packages and libraries /--

entity pm_16 is
      generic (file_name: string);
      port(receiv_16s: inout token;
            out_s: inout token);
end pm_16;

architecture behave_pm_16 of pm_16 is

begin
decoder: process

begin
   t_tmp:= t_tmp + 1;
   if (t_tmp < 16) then
       syn_receive(receiv_16s, t_temp, 99999 ns);
       temp:= t_temp.color.data2;

   --/ processing 16 bits information /--
       buf_reg:= buf_reg(1 to 15) & temp;
       t_back:=  syndrom(9);
       t_forward:= temp;
       syndrom:= r_decode_sydrom(syndrom,t_back,t_forward);
   else
  --/ sending off 16 bits information /--
       t_temp.color.data1:= bv_to_unsignint(buf_reg);
       t_temp.color.condi:= TRUE;
       syn_transmit(out_s, t_temp, 99999 ns);

  --/ sending off 10 bits syndrom /--
       t_temp.color.data1:= bv_to_unsignint(syndrom);
       t_temp.color.condi:= TRUE;
       syn_transmit(out_s, t_temp, 99999 ns);

  --/ initializing the pm_16 again /--
       t_tmp:= -1;
       syndrom:= (others => '0');
       buf_reg:= (others => '0');
   end if;

 --/ counting invoking time /--
    write(l, t_times, right, 1);
    t_times:= not t_times;
    if t_count >= 29 then
       write(l, NOW, right, 15);
       writeline(datafile,l);
       t_count:= -1;
    end if;
    t_count:= t_count + 1;

end process;
```

end behave_pm_16;

- **CDFG**



- **DFGs**



- **Hardware Cost and Performance for the Main DFG**

Node	Priority	ASAP	Schd_1	Schd_2	Schd_3	Schd_4
V1	1	1	1	1	1	2
V2	4	1	1	1	1	1
V3	1	1	1	1	2	3
V4	1	2	2	2	3	6
V5	1	2	2	3	4	7
V6	1	2	2	3	4	8
V7	1	2	2	2	3	4
V8	1	3	3	3	4	5
V9	1	4	4	4	5	9
V10	1	5	5	5	6	10
V11	1	6	6	6	7	11
V12	1	7	7	7	8	12
V13	1	1	1	2	2	4
V14	1	1	2	2	3	5
V15	1	8	8	8	9	13
V16	1	9	9	9	10	14
V17	1	10	10	10	11	15
Total Steps	N/A	**10**	**10**	**10**	**11**	**15**
Total Compnts	N/A	5 "XOR" 1 "&"	4 "XOR" 1 "&"	3 "XOR" 1 "&"	2 "XOR" 1 "&"	1 "XOR" 1 "&"

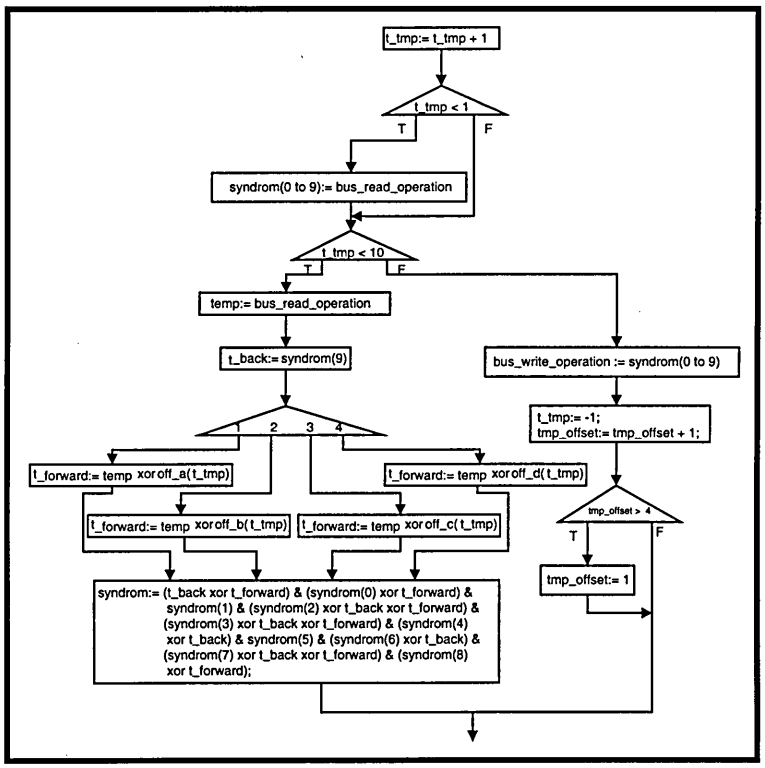
2. PCW_10_A/B

- **VHDL Program**

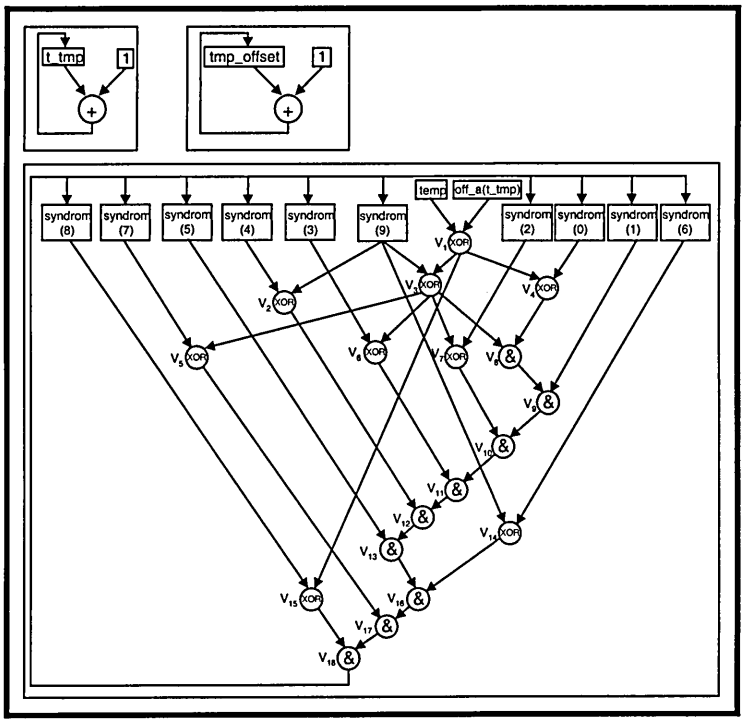
```
--/ here are definitions for VHDL packages and libraries /--

entity pcw_10 is
      generic (file_name: string);
      port(receiv_10s: inout token;
           receiv_syndrom: inout token;
           out_s: inout token);
end pcw_10;

architecture behave_pcw_10 of pcw_10 is

begin
decoder: process

--/ here are definitions for variables and signals /--

begin
   t_tmp:= t_tmp + 1;
   if (t_tmp < 1) then
      syn_receive(receiv_syndrom, t_temp, 99999 ns);
      tmp:= t_temp.color.data1;
      syndrom:= unsignint_to_bv(tmp, 10);
   end if;
   if (t_tmp < 10) then
      syn_receive(receiv_10s, t_temp, 99999 ns);
      temp:= t_temp.color.data2;
```

```
    --/ processing checkword /--
        t_back:= syndrom(9);
        case tmp_offset is
            when 1 =>
                t_forward:= temp xor off_a(t_tmp);
            when 2 =>
                t_forward:= temp xor off_b(t_tmp);
            when 3 =>
                t_forward:= temp xor off_c(t_tmp);
            when others =>
                t_forward:= temp xor off_d(t_tmp);
        end case;
        syndrom:= r_decode_sydrom(syndrom,t_back,t_forward);
    else
    --/ sending off 10 bits syndrom /--
        t_temp.color.data1:= bv_to_unsignint(syndrom);
        t_temp.color.condi:= TRUE;
        syn_transmit(out_s, t_temp, 99999 ns);

    --/ initializing the pcw_10 again /--
        t_tmp:= -1;
        tmp_offset:= tmp_offset + 1;
        if tmp_offset > 4 then
          tmp_offset:= 1;
        end if;
    end if;

--/ counting invoking time /--
    write(l, t_times, right, 1);
    t_times:= not t_times;
    if t_count >= 29 then
       write(l, NOW, right, 15);
       writeline(datafile,l);
       t_count:= -1;
    end if;
    t_count:= t_count + 1;

end process;

end behave_pcw_10;
```

- **CDFG**


t_tmp:= t_tmp + 1
t_tmp < 1
T
F
syndrom(0 to 9):= bus_read_operation
t_tmp < 10
T
F
temp:= bus_read_operation
t_back:= syndrom(9)
bus_write_operation := syndrom(0 to 9)
t_tmp:= -1;
tmp_offset:= tmp_offset + 1;
1 2 3 4
t_forward:= temp xor off_a(t_tmp)
t_forward:= temp xor off_d(t_tmp)
t_forward:= temp xor off_b(t_tmp)
t_forward:= temp xor off_c(t_tmp)
tmp_offset > 4
T
F
tmp_offset:= 1
syndrom:= (t_back xor t_forward) & (syndrom(0) xor t_forward) &
syndrom(1) & (syndrom(2) xor t_back xor t_forward) &
(syndrom(3) xor t_back xor t_forward) & (syndrom(4)
xor t_back) & syndrom(5) & (syndrom(6) xor t_back) &
(syndrom(7) xor t_back xor t_forward) & (syndrom(8)
xor t_forward);

- **DFGs**


t_tmp
1
tmp_offset
1
temp
off_a(t_tmp)
syndrom (8)
syndrom (7)
syndrom (5)
syndrom (4)
syndrom (3)
syndrom (9)
syndrom (2)
syndrom (0)
syndrom (1)
syndrom (6)
XOR
&

- **Hardware Cost and Performance for the Main DFG**

Node	Priority	ASAP	Schd_1	Schd_2
V1	3	1	1	1
V2	1	1	1	3
V3	4	2	2	2
V4	1	2	3	5
V5	1	3	4	7
V6	1	3	4	8
V7	1	3	5	9
V8	1	3	4	6
V9	1	4	5	7
V10	1	5	6	10
V11	1	6	7	11
V12	1	7	8	12
V13	1	8	9	13
V14	1	1	2	4
V15	1	2	3	6
V16	1	9	10	14
V17	1	10	11	15
V18	1	11	12	16
Total Steps	N/A	**11**	**12**	**16**
Total Compnts	N/A	3 "XOR" 1 "&"	2 "XOR" 1 "&"	1 "XOR" 1 "&"

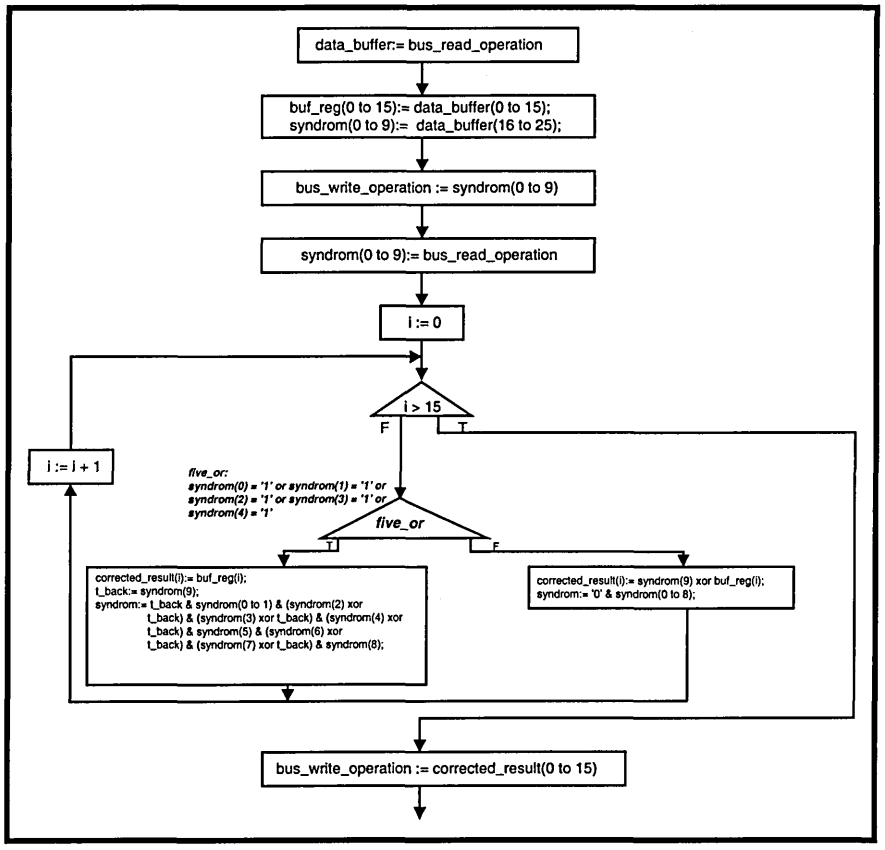
3. Correctr A/B

- **VHDL Program**

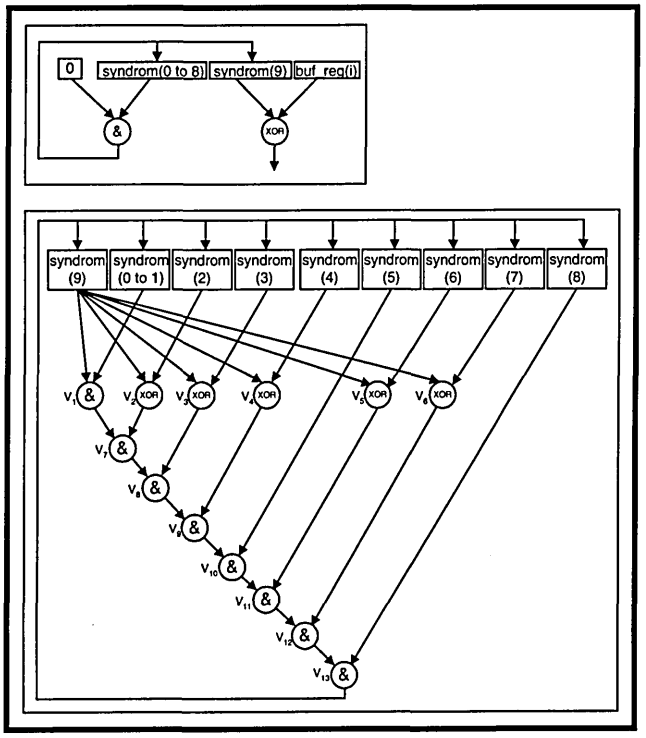
```
--/ here are definitions for VHDL packages and libraries /--

entity corrector is
      generic (file_name: string);
      port(in_pm: inout token; in_pcw: inout token;
          out_pcw: inout token; corrected: inout token);
end corrector;

architecture behave_corrector of corrector is

begin
corrector: process

--/ here are definitions for variables and signals /--

begin
--/ receiving 16 bits information /--
   syn_receive(in_pm, t_temp, 99999 ns);
   tmp:= t_temp.color.data1;
   buf_reg:= unsignint_to_bv(tmp, 16);

--/ receiving 10 bits syndrom /--
   syn_receive(in_pm, t_temp, 99999 ns);
   -- tmp:= t_temp.color.data1;

--/ sending off the 10 bits syndrom /--
```

```
  --  t_temp.color.data1:= tmp;
    t_temp.color.condi:= TRUE;
    syn_transmit(out_pcw, t_temp, 99999 ns);

--/ receiving 10 bits check word /--
    syn_receive(in_pcw, t_temp, 99999 ns);
    tmp:= t_temp.color.data1;
    syndrom:= unsignint_to_bv(tmp, 10);

--/ the correction goes here /--
    for i in 0 to 15 loop
       if five_nor(syndrom(0 to 4)) = '1' then
         corrected_result(i):= buf_reg(i);
         t_back:= syndrom(9);
         syndrom:= r_encode_sydrom(syndrom, t_back);
       else
         corrected_result(i):= syndrom(9) xor buf_reg(i);
         syndrom:= '0' & syndrom(0 to 8);
       end if;
    end loop;

--/ sending off the corrected 16 bits information /--
    t_temp.color.data1:= bv_to_unsignint(corrected_result);
    t_temp.color.condi:= TRUE;
    syn_transmit(corrected, t_temp, 99999 ns);

 --/ counting invoking time /--
     write(l, t_times, right, 1);
     t_times:= not t_times;
     if t_count >= 29 then
       write(l, NOW, right, 15);
       writeline(datafile,l);
       t_count:= -1;
     end if;
     t_count:= t_count + 1;

end process;

end behave_corrector;
```

- **CDFG**


data_buffer:= bus_read_operation
buf_reg(0 to 15):= data_buffer(0 to 15);
syndrom(0 to 9):= data_buffer(16 to 25);
bus_write_operation := syndrom(0 to 9)
syndrom(0 to 9):= bus_read_operation
i := 0
i > 15
F
T
i := i + 1
five_or:
syndrom(0) = '1' or syndrom(1) = '1' or
syndrom(2) = '1' or syndrom(3) = '1' or
syndrom(4) = '1'
five_or
T
F
corrected_result(i):= buf_reg(i);
t_back:= syndrom(9);
syndrom:= t_back & syndrom(0 to 1) & (syndrom(2) xor
t_back) & (syndrom(3) xor t_back) & (syndrom(4) xor
t_back) & syndrom(5) & (syndrom(6) xor
t_back) & (syndrom(7) xor t_back) & syndrom(8);
corrected_result(i):= syndrom(9) xor buf_reg(i);
syndrom:= '0' & syndrom(0 to 8);
bus_write_operation := corrected_result(0 to 15)

- **DFGs**


0
syndrom(0 to 8)
syndrom(9)
buf_reg(i)
&
XOR
syndrom (9)
syndrom (0 to 1)
syndrom (2)
syndrom (3)
syndrom (4)
syndrom (5)
syndrom (6)
syndrom (7)
syndrom (8)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13

- **Hardware Cost and Performance for the Main DFG**

Node	Priority	ASAP	Schd_1	Schd_2	Schd_3	Schd_4
V1	1	1	1	1	1	1
V2	1	1	1	1	1	2
V3	1	1	1	1	2	3
V4	1	1	1	2	2	4
V5	1	1	2	2	3	5
V6	1	1	2	2	3	6
V7	1	2	2	2	2	3
V8	1	3	3	3	3	4
V9	1	4	4	4	4	5
V10	1	5	5	5	5	6
V11	1	6	6	6	6	7
V12	1	7	7	7	7	8
V13	1	8	8	8	8	9
Total Steps	N/A	**8**	**8**	**8**	**8**	**9**
Total Compnts	N/A	5 "XOR" 1 "&"	4 "XOR" 1 "&"	3 "XOR" 1 "&"	2 "XOR" 1 "&"	1 "XOR" 1 "&"

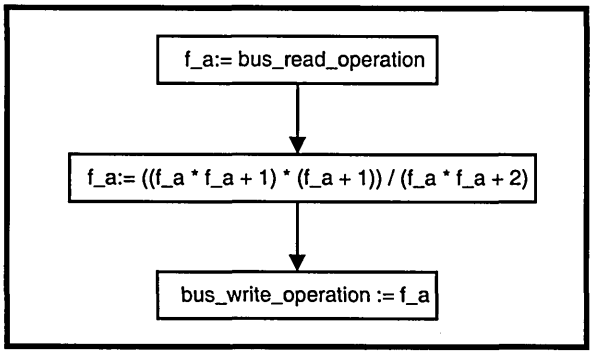
4. $F_{\_A}$

- **VHDL Program**

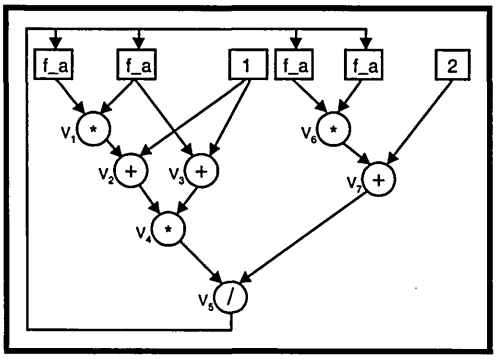
```
--/ here are definitions for VHDL packages and libraries /--

entity f_a_block is
      port(in_16:  inout token;
           out_16: inout token);
end f_a_block;

architecture behave_f_a_block of f_a_block is

begin
f_block: process

--/ here are definitions for variables and signals /--

begin
    syn_receive(in_16, temp_in, 99999 ns);
    f_a:= integer(temp_in.color.data1);

    f_a:= ((f_a * f_a + 1) * (f_a + 1)) / (f_a * f_a + 2);

    temp_out.color.data1:= unsigned_int(f_a);
    temp_out.color.condi:= true;
    syn_transmit(out_16, temp_out, 99999 ns);

 --/ counting invoking time /--
    write(l, t_times, right, 1);
    t_times:= not t_times;
    if t_count >= 29 then
      write(l, NOW, right, 15);
```

```
      writeline(datafile,l);
      t_count:= -1;
    end if;
    t_count:= t_count + 1;

end process;

end behave_f_a_block;
```

- **CDFG**



- **DFGs**



- **Hardware Cost and Performance for the Main DFG**

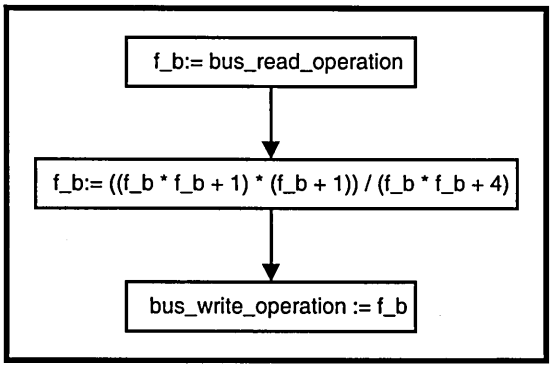
Node	Priority	ASAP	Schd_1	Schd_2	Schd_3
V1	1	1	1	1	1
V2	1	2	2	2	2
V3	1	1	1	1	1
V4	1	3	3	3	3
V5	1	4	4	4	4
V6	1	1	2	1	2
V7	1	2	3	3	3
Total Steps	N/A	**4**	**4**	**4**	**4**
Total Compnts	N/A	2 "*", 2 "+" 1 "/"	1 "*", 2 "+" 1 "/"	2 "*", 1 "+" 1 "/"	1 "*", 1 "+" 1 "/"

5. $F_{\_B}$

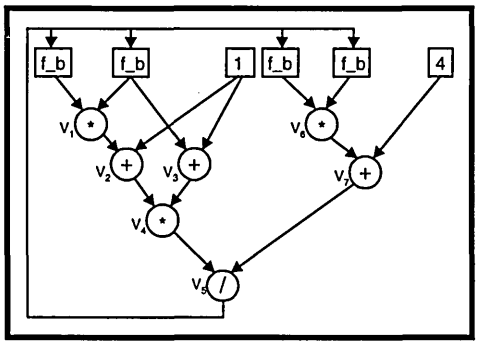
- **VHDL Program**

```
entity f_b_block is
      port(in_16:  inout token;
           out_16: inout token);
end f_b_block;

architecture behave_f_b_block of f_b_block is

begin
f_block: process
begin

    syn_receive(in_16, temp_in, 99999 ns);
    f_b:= integer(temp_in.color.data1);

    f_b:= ((f_b * f_b + 1) * (f_b + 1)) / (f_b * f_b + 4);

    temp_out.color.data1:= unsigned_int(f_b);
    temp_out.color.condi:= true;
    syn_transmit(out_16, temp_out, 99999 ns);

 --/ counting invoking time /--
    write(l, t_times, right, 1);
    t_times:= not t_times;
    if t_count >= 29 then
       write(l, NOW, right, 15);
       writeline(datafile,l);
       t_count:= -1;
    end if;
    t_count:= t_count + 1;

end process;

end behave_f_b_block;
```

- **CDFG**



- **DFGs**



- **Hardware Cost and Performance for the Main DFG**

Similar as F_A's.

Appendix J

1. Co-simulation Program (*one bus layer*)

```
LIBRARY communication;
USE communication.ESSENTIAL_DEFINITIONS.ALL;
USE communication.token_definition.ALL;
USE communication.token_passing.ALL;
USE communication.par_vhdl_conversion.ALL;
USE communication.RDS_UTILITIES.ALL;
USE std.textio.ALL;

ENTITY rds_matrix_1 IS
END rds_matrix_1;

ARCHITECTURE behave_rds_matrix_1 OF rds_matrix_1 IS

COMPONENT bus_arbiter
        PORT (
          -- arbitrat_bus --
            br: IN word;
            bg: OUT word:= word_high;
            bbsy: IN BIT
           );
END COMPONENT;

COMPONENT synchro_same
        PORT (clk: IN BIT;
          -- address_bus signals --
            atb: IN word;
          -- data_bus signals --
            dtb: INOUT or_lword_res BUS;
          -- control_bus signals --
            as, ds, rw, sd, sa: IN BIT;
            dtack, ready: INOUT and_bit_res BUS := '1'
           );
END COMPONENT;

COMPONENT asynchro_same
        GENERIC (que_id: word);
        PORT (clk: IN BIT;
          -- address_bus signals
            atb: IN word;
            segmt: IN word;
          -- data_bus signals
            dtb: INOUT or_lword_res BUS;
          -- control_bus signals
            as, ds, rw, sd, sa: IN BIT;
            dtack, ready: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT clk_gen
        GENERIC (delay_length: TIME);
        PORT    (clk: inout BIT);
END COMPONENT;

COMPONENT transmit_a
```

```
        PORT(sent_16s: inout token;
             sent_10s: inout token);
END COMPONENT;

COMPONENT transmit_b
        PORT(sent_16s: inout token;
             sent_10s: inout token);
END COMPONENT;

COMPONENT pcw_10_h
        GENERIC (b_id:      word;
                 chanl_in:  word;
                 chanl_out: word);
        PORT    (clk: IN BIT;
                 data_token: INOUT token;
                -- address bus --
                 atb_glob: INOUT or_word_res BUS;
                 segmt_glob: INOUT or_word_res BUS;
                -- data bus --
                 dtb_glob: INOUT or_lword_res BUS;
                -- control bus --
                 as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
                 dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
                 rw_glob: INOUT or_bit_res BUS;
                 sd_glob: INOUT or_bit_res BUS;
                 sa_glob: INOUT or_bit_res BUS;
                -- arbitration bus --
                 br_glob: INOUT and_word_res BUS := word_high;
                 bg_glob: IN word;
                 bbsy_glob: INOUT and_bit_res BUS:= '1'
                );
END COMPONENT;

COMPONENT pm_16_h
        GENERIC (b_id:      word;
                 chanl_out: word );
        PORT    (clk: IN BIT;
                 data_token: INOUT token;
                -- address bus --
                 atb_glob: INOUT or_word_res BUS;
                 segmt_glob: INOUT or_word_res BUS;
                -- data bus --
                 dtb_glob: INOUT or_lword_res BUS;
                -- control bus --
                 as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
                 dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
                 rw_glob: INOUT or_bit_res BUS;
                 sd_glob: INOUT or_bit_res BUS;
                 sa_glob: INOUT or_bit_res BUS;
                -- arbitration bus --
                 br_glob: INOUT and_word_res BUS := word_high;
                 bg_glob: IN word;
                 bbsy_glob: INOUT and_bit_res BUS:= '1'
                );
END COMPONENT;

COMPONENT corrector_h
        GENERIC (b_id:      word;
                 chanl_in1: word;
                 chanl_in2: word;
```

```
            chanl_out1: word;
            chanl_out2: word);
        PORT    (clk: IN BIT;
            -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
            -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
            -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
            );
END COMPONENT;

COMPONENT f_a_s
        GENERIC (b_id:     word;
            chanl_in:  word;
            chanl_out: word);
        PORT    (clk: IN BIT;
            data_token: inout token;
            -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
            -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
            -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
            );
END COMPONENT;

COMPONENT f_b_h
        GENERIC (b_id:     word;
            chanl_in:  word;
            chanl_out: word);
        PORT    (clk: IN BIT;
            -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
            -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
```

```
            sa_glob: INOUT or_bit_res BUS;
          -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
          );
END COMPONENT;

COMPONENT control_block
      GENERIC (b_id:    word;
            chanl_in1: word;
            chanl_in2: word;
            chanl_out: word);
      PORT   (clk: IN BIT;
            data_token: inout token;
          -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
          -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
          -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
          -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
          );
END COMPONENT;

COMPONENT store
      GENERIC (b_id:    word;
            chanl_in: word);
      PORT   (clk: IN BIT;
          -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
          -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
          -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
          -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
          );
END COMPONENT;

--- /// Global Bus Signals /// ---
-- adress bus --
SIGNAL atb: or_word_res BUS;
SIGNAL segmt: or_word_res BUS;
-- data bus --
```

```
SIGNAL dtb: or_lword_res BUS;
-- control bus --
SIGNAL as, ds: and_bit_res BUS := '1';
SIGNAL dtack, ready: and_bit_res BUS := '1';
SIGNAL rw: or_bit_res BUS;
SIGNAL sd: or_bit_res BUS;
SIGNAL sa: or_bit_res BUS;
-- arbitration bus --
SIGNAL br: and_word_res BUS:= word_high;
SIGNAL bg: word:= word_high;
SIGNAL bbsy: and_bit_res BUS:= '1';

SIGNAL transmit_11, transmit_12: token_res;
SIGNAL transmit_21, transmit_22: token_res;
SIGNAL fa_control: token_res;

---/// other signals ///---
SIGNAL clk_syn, clk_asyn, clk_corrector1, clk_corrector2,
     clk_pm1, clk_pcw1, clk_pm2, clk_pcw2, clk_contrl,
     clk_f_a, clk_f_b, clk_storage: BIT:='0';

BEGIN

syn_ck_1: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_syn);

asyn_clk: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_asyn);

pm_clk1: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_pm1);

pm_clk2: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_pm2);

pcw_clk1: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_pcw1);

pcw_clk2: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_pcw2);

corect_1: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_corrector1);

corect_2: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_corrector2);

f_a_clk: clk_gen
       GENERIC MAP(50 ns)
       PORT    MAP(clk_f_a);

f_b_clk: clk_gen
```

```
        GENERIC MAP(25 ns)
        PORT    MAP(clk_f_b);

contl_ck: clk_gen
        GENERIC MAP(50 ns)
        PORT    MAP(clk_contrl);

stor_clk: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_storage);

bus_a1: bus_arbiter
      PORT MAP (br, bg, bbsy);

synchro: synchro_same
       PORT MAP (clk_syn, atb, dtb, as, ds, rw, sd, sa,
             dtack, ready);

asynchro: asynchro_same
        GENERIC MAP ("1111111111111101")
        PORT    MAP (clk_asyn, atb, segmt, dtb, as, ds, rw,
                sd, sa, dtack, ready);

-- /// Individual Modules in the System /// --

Transmiter_A: transmit_a
          PORT MAP (transmit_11, transmit_12);

Transmiter_B: transmit_b
          PORT MAP (transmit_21, transmit_22);

PM_16_A: pm_16_h
       GENERIC MAP ("1111111111111011", "1111111111111101")
       PORT    MAP (clk_pm1, transmit_11, atb, segmt, dtb, as, ds,
              dtack, ready, rw, sd, sa, br, bg, bbsy);

PM_16_B: pm_16_h
       GENERIC MAP ("1111110111111111", "1111111011111111")
       PORT    MAP (clk_pm2, transmit_21, atb, segmt, dtb, as, ds,
              dtack, ready, rw, sd, sa, br, bg, bbsy);

PCW_10_A: pcw_10_h
        GENERIC MAP ("1111111111110111", "1111111111111011",
                "1111111111110111")
        PORT    MAP (clk_pcw1, transmit_12, atb, segmt, dtb, as, ds,
                dtack, ready, rw, sd, sa, br, bg, bbsy);

PCW_10_B: pcw_10_h
        GENERIC MAP ("1111101111111111", "1111110111111111",
                "1111101111111111")
        PORT    MAP (clk_pcw2, transmit_22, atb, segmt, dtb, as, ds,
                dtack, ready, rw, sd, sa, br, bg, bbsy);

corrector_A: corrector_h
          GENERIC MAP ("1111111111101111", "1111111111111101",
                  "1111111111110111", "1111111111111011",
                  "1111111111101111")
          PORT    MAP (clk_corrector1, atb, segmt, dtb, as, ds,
                  dtack, ready, rw, sd, sa,
                  br, bg, bbsy);
```

```
corrector_b: corrector_h
        GENERIC MAP ("1111111011111111", "1111111011111111",
                "1111101111111111", "1111110111111111",
                "1111111101111111")
        PORT    MAP (clk_corrector2, atb, segmt, dtb, as, ds,
                dtack, ready, rw, sd, sa,
                br, bg, bbsy);

f_a: f_a_s
    GENERIC MAP ("1111111111011111", "1111111111101111",
            "1111111111011111")
    PORT MAP (clk_f_a, fa_control, atb, segmt, dtb, as, ds,
            dtack, ready, rw, sd, sa,
            br, bg, bbsy);

f_b: f_b_h
    GENERIC MAP ("1111111101111111", "1111111101111111",
            "1111111110111111")
    PORT MAP (clk_f_b, atb, segmt, dtb, as, ds,
            dtack, ready, rw, sd, sa,
            br, bg, bbsy);

control: control_block
       GENERIC MAP ("1111111110111111", "1111111111011111",
               "1111111110111111", "1111111111111101")
       PORT    MAP (clk_contrl, fa_control, atb, segmt, dtb, as, ds,
               dtack, ready, rw, sd, sa,
               br, bg, bbsy);

storage: store
       GENERIC MAP ("1111011111111111", "1111111111111101")
       PORT    MAP (clk_storage, atb, segmt, dtb, as, ds,
               dtack, ready, rw, sd, sa,
               br, bg, bbsy);

END behave_rds_matrix_1;
```

2. Co-simulation Program (*two bus layers*)

```
LIBRARY communication;
USE communication.ESSENTIAL_DEFINITIONS.ALL;
USE communication.token_definition.ALL;
USE communication.token_passing.ALL;
USE communication.par_vhdl_conversion.ALL;
USE communication.RDS_UTILITIES.ALL;
USE std.textio.ALL;

ENTITY rds_matrix_2 IS
END rds_matrix_2;

ARCHITECTURE behave_rds_matrix_2 OF rds_matrix_2 IS

COMPONENT bus_arbiter
        PORT (
          -- arbitrat_bus --
            br: IN word;
            bg: OUT word:= word_high;
            bbsy: IN BIT
           );
```

```
END COMPONENT;

COMPONENT synchro_same
        PORT (clk: IN BIT;
          -- address_bus signals --
            atb: IN word;
          -- data_bus signals --
            dtb: INOUT or_lword_res BUS;
          -- control_bus signals --
            as, ds, rw, sd, sa: IN BIT;
            dtack, ready: INOUT and_bit_res BUS := '1'
           );
END COMPONENT;

COMPONENT asynchro_same
        GENERIC (que_id: word);
        PORT (clk: IN BIT;
          -- address_bus signals
            atb: IN word;
            segmt: IN word;
          -- data_bus signals
            dtb: INOUT or_lword_res BUS;
          -- control_bus signals
            as, ds, rw, sd, sa: IN BIT;
            dtack, ready: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT synchro_differ
        GENERIC (arbitra_id1: word;
                 arbitra_id2: word);
        PORT    (clk: IN BIT;
            -- No.1 bus signals --
            -- address bus
              atb1: INOUT or_word_res BUS;
              segmt1: INOUT or_word_res BUS;
            -- data bus
              dtb1: INOUT or_lword_res BUS;
            -- control bus
              as1, ds1: INOUT and_bit_res BUS;
              rw1, sd1, sa1: INOUT or_bit_res BUS;
              dtack1, ready1: INOUT and_bit_res BUS := '1';
            -- arbitration bus
              br1: INOUT and_word_res BUS:= word_high;
              bg1: IN word;
              bbsy1: INOUT and_bit_res BUS:= '1';

            -- No.2 bus signals --
            -- address bus
              atb2: INOUT or_word_res BUS;
              segmt2: INOUT or_word_res BUS;
            -- data bus
              dtb2: INOUT or_lword_res BUS;
            -- control bus
              as2, ds2: INOUT and_bit_res BUS;
              rw2, sd2, sa2: INOUT or_bit_res BUS;
              dtack2, ready2: INOUT and_bit_res BUS := '1';
            -- arbitration bus
              br2: INOUT and_word_res BUS:= word_high;
              bg2: IN word;
```

```
            bbsy2: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT clk_gen
      GENERIC (delay_length: TIME);
      PORT   (clk: inout BIT);
END COMPONENT;

COMPONENT transmit_a
      PORT(sent_16s: inout token;
         sent_10s: inout token);
END COMPONENT;

COMPONENT transmit_b
      PORT(sent_16s: inout token;
         sent_10s: inout token);
END COMPONENT;

COMPONENT pcw_10_h
      GENERIC (b_id:     word;
            chanl_in:  word;
            chanl_out: word);
      PORT   (clk: IN BIT;
            data_token: INOUT token;
         -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
         -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
         -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
         -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT pm_16_h
      GENERIC (b_id:     word;
            chanl_out: word );
      PORT   (clk: IN BIT;
            data_token: INOUT token;
         -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
         -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
         -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
         -- arbitration bus --
```

```
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT corrector_h
      GENERIC (b_id:      word;
            chanl_in1:  word;
            chanl_in2:  word;
            chanl_out1: word;
            chanl_out2: word);
      PORT    (clk: IN BIT;
          -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
          -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
          -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
          -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT f_a_s
      GENERIC (b_id:     word;
            chanl_in:  word;
            chanl_out: word);
      PORT    (clk: IN BIT;
            data_token: inout token;
          -- address bus --
            atb_glob: INOUT or_word_res BUS;
            segmt_glob: INOUT or_word_res BUS;
          -- data bus --
            dtb_glob: INOUT or_lword_res BUS;
          -- control bus --
            as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
            dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
            rw_glob: INOUT or_bit_res BUS;
            sd_glob: INOUT or_bit_res BUS;
            sa_glob: INOUT or_bit_res BUS;
          -- arbitration bus --
            br_glob: INOUT and_word_res BUS := word_high;
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

COMPONENT f_b_h
      GENERIC (b_id:     word;
            chanl_in:  word;
            chanl_out: word);
      PORT    (clk: IN BIT;
```

```
            -- address bus --
              atb_glob: INOUT or_word_res BUS;
              segmt_glob: INOUT or_word_res BUS;
            -- data bus --
              dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
              as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
              dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
              rw_glob: INOUT or_bit_res BUS;
              sd_glob: INOUT or_bit_res BUS;
              sa_glob: INOUT or_bit_res BUS;
            -- arbitration bus --
              br_glob: INOUT and_word_res BUS := word_high;
              bg_glob: IN word;
              bbsy_glob: INOUT and_bit_res BUS:= '1'
             );
END COMPONENT;

COMPONENT control_block
      GENERIC (b_id:     word;
              chanl_in1: word;
              chanl_in2: word;
              chanl_out: word);
      PORT    (clk: IN BIT;
              data_token: inout token;
            -- address bus --
              atb_glob: INOUT or_word_res BUS;
              segmt_glob: INOUT or_word_res BUS;
            -- data bus --
              dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
              as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
              dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
              rw_glob: INOUT or_bit_res BUS;
              sd_glob: INOUT or_bit_res BUS;
              sa_glob: INOUT or_bit_res BUS;
            -- arbitration bus --
              br_glob: INOUT and_word_res BUS := word_high;
              bg_glob: IN word;
              bbsy_glob: INOUT and_bit_res BUS:= '1'
             );
END COMPONENT;

COMPONENT store
      GENERIC (b_id:     word;
              chanl_in: word);
      PORT    (clk: IN BIT;
            -- address bus --
              atb_glob: INOUT or_word_res BUS;
              segmt_glob: INOUT or_word_res BUS;
            -- data bus --
              dtb_glob: INOUT or_lword_res BUS;
            -- control bus --
              as_glob, ds_glob: INOUT and_bit_res BUS:= '1';
              dtack_glob, ready_glob: INOUT and_bit_res BUS:= '1';
              rw_glob: INOUT or_bit_res BUS;
              sd_glob: INOUT or_bit_res BUS;
              sa_glob: INOUT or_bit_res BUS;
            -- arbitration bus --
              br_glob: INOUT and_word_res BUS := word_high;
```

```
            bg_glob: IN word;
            bbsy_glob: INOUT and_bit_res BUS:= '1'
           );
END COMPONENT;

--- /// Global Bus1 Signals /// ---

-- adress bus1 --
SIGNAL atb1: or_word_res BUS;
SIGNAL segmt1: or_word_res BUS;
-- data bus --
SIGNAL dtb1: or_lword_res BUS;
-- control bus1 --
SIGNAL as1, ds1: and_bit_res BUS := '1';
SIGNAL dtack1, ready1: and_bit_res BUS := '1';
SIGNAL rw1: or_bit_res BUS;
SIGNAL sd1: or_bit_res BUS;
SIGNAL sa1: or_bit_res BUS;
-- arbitration bus1 --
SIGNAL br1: and_word_res BUS:= word_high;
SIGNAL bg1: word:= word_high;
SIGNAL bbsy1: and_bit_res BUS:= '1';

--- /// Global Bus2 Signals /// ---

-- adress bus2 --
SIGNAL atb2: or_word_res BUS;
SIGNAL segmt2: or_word_res BUS;
-- data bus2 --
SIGNAL dtb2: or_lword_res BUS;
-- control bus2 --
SIGNAL as2, ds2: and_bit_res BUS := '1';
SIGNAL dtack2, ready2: and_bit_res BUS := '1';
SIGNAL rw2: or_bit_res BUS;
SIGNAL sd2: or_bit_res BUS;
SIGNAL sa2: or_bit_res BUS;
-- arbitration bus2 --
SIGNAL br2: and_word_res BUS:= word_high;
SIGNAL bg2: word:= word_high;
SIGNAL bbsy2: and_bit_res BUS:= '1';

SIGNAL transmit_11, transmit_12: token_res;
SIGNAL transmit_21, transmit_22: token_res;
SIGNAL fa_control: token_res;

---/// other signals ///---
SIGNAL clk_syn_same1, clk_syn_same2, clk_syn_differ, clk_asyn,
     clk_pm1, clk_pcw1, clk_pm2, clk_pcw2, clk_f_a, clk_f_b,
     clk_corrector1, clk_corrector2, clk_contrl, clk_storage: BIT:='0';

BEGIN

syn_ck_1: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_syn_same1);

syn_ck_2: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_syn_same2);
```

```
syn_ck_d: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_syn_differ);

asyn_clk: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_asyn);

pm_clk1: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_pm1);

pm_clk2: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_pm2);

pcw_clk1: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_pcw1);

pcw_clk2: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_pcw2);

corect_1: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_corrector1);

corect_2: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_corrector2);

f_a_clk: clk_gen
       GENERIC MAP(50 ns)
       PORT    MAP(clk_f_a);

f_b_clk: clk_gen
       GENERIC MAP(25 ns)
       PORT    MAP(clk_f_b);

contl_ck: clk_gen
        GENERIC MAP(50 ns)
        PORT    MAP(clk_contrl);

stor_clk: clk_gen
        GENERIC MAP(25 ns)
        PORT    MAP(clk_storage);

bus_abi1: bus_arbiter
        PORT MAP (br1, bg1, bbsy1);

bus_abi2: bus_arbiter
        PORT MAP (br2, bg2, bbsy2);

s_same_1: synchro_same
        PORT MAP (clk_syn_same1, atb1, dtb1, as1, ds1, rw1,
                sd1, sa1, dtack1, ready1);

s_same_2: synchro_same
        PORT MAP (clk_syn_same2, atb2, dtb2, as2, ds2, rw2,
```

```
                sd2, sa2, dtack2, ready2);

 s_difer: synchro_differ
        GENERIC MAP ("1110111111111111", "1110111111111111")
        PORT    MAP (clk_syn_differ,
                atb1, segmt1, dtb1, as1, ds1, rw1, sd1, sa1,
                dtack1, ready1, br1, bg1, bbsy1,
                atb2, segmt2, dtb2, as2, ds2, rw2, sd2, sa2,
                dtack2, ready2, br2, bg2, bbsy2);

 asynchro1: asynchro_same
         GENERIC MAP ("1111111111111101")
         PORT    MAP (clk_asyn, atb2, segmt2, dtb2, as2, ds2, rw2,
                  sd2, sa2, dtack2, ready2);

 -- /// Individual Modules in the System /// --

 Transmiter_A: transmit_a
          PORT MAP (transmit_11, transmit_12);

 Transmiter_B: transmit_b
          PORT MAP (transmit_21, transmit_22);

 PM_16_A: pm_16_h
       GENERIC MAP ("1111111111111011", "1111111111111101")
       PORT    MAP (clk_pm1, transmit_11, atb1, segmt1,
                dtb1, as1, ds1, dtack1, ready1, rw1,
                sd1, sa1, br1, bg1, bbsy1);

 PM_16_B: pm_16_h
       GENERIC MAP ("1111110111111111", "1111111011111111")
       PORT    MAP (clk_pm2, transmit_21, atb2, segmt2,
                dtb2, as2, ds2, dtack2, ready2, rw2,
                sd2, sa2, br2, bg2, bbsy2);

 PCW_10_A: pcw_10_h
        GENERIC MAP ("1111111111110111", "1111111111111011",
                 "1111111111110111")
        PORT    MAP (clk_pcw1, transmit_12, atb1, segmt1,
                dtb1, as1, ds1, dtack1, ready1, rw1,
                sd1, sa1, br1, bg1, bbsy1);

 PCW_10_B: pcw_10_h
       GENERIC MAP ("1111101111111111", "1111110111111111",
                "1111101111111111")
       PORT    MAP (clk_pcw2, transmit_22, atb2, segmt2,
                dtb2, as2, ds2, dtack2, ready2, rw2,
                sd2, sa2, br2, bg2, bbsy2);

 corect_A: corrector_h
       GENERIC MAP ("1111111111101111", "1111111111111101",
                "1111111111110111", "1111111111111011",
                "1111111111101111")
       PORT    MAP (clk_corrector1, atb1, segmt1,
                dtb1, as1, ds1, dtack1, ready1, rw1,
                sd1, sa1, br1, bg1, bbsy1);

 corect_B: corrector_h
       GENERIC MAP ("1111111011111111", "1111111011111111",
                "1111101111111111", "1111110111111111",
```

```
                "1111111101111111")
        PORT    MAP (clk_corrector2, atb2, segmt2,
                dtb2, as2, ds2, dtack2, ready2, rw2,
                sd2, sa2, br2, bg2, bbsy2);

f_a: f_a_s
    GENERIC MAP ("1111111111011111", "1111111111101111",
            "1111111111011111")
    PORT    MAP (clk_f_a, fa_control, atb1, segmt1, dtb1, as1, ds1,
            dtack1, ready1, rw1, sd1, sa1, br1, bg1, bbsy1);

f_b: f_b_h
    GENERIC MAP ("1111111101111111", "1111111101111111",
            "1111111110111111")
    PORT    MAP (clk_f_b, atb2, segmt2, dtb2, as2, ds2,
            dtack2, ready2, rw2, sd2, sa2, br2, bg2, bbsy2);

contl: control_block
     GENERIC MAP ("1111111110111111", "1111111111011111",
             "1111111110111111", "1111111111111101")
     PORT    MAP (clk_contrl, fa_control, atb2, segmt2, dtb2, as2, ds2,
             dtack2, ready2, rw2, sd2, sa2, br2, bg2, bbsy2);

stor: store
    GENERIC MAP ("1111011111111111", "1111111111111101")
    PORT    MAP (clk_storage, atb2, segmt2, dtb2, as2, ds2,
            dtack2, ready2, rw2, sd2, sa2, br2, bg2, bbsy2);

END behave_rds_matrix_2;
```